1989

A computational comparison of the primal simplex and relaxation algorithms for solving minimum cost flow networks.

Sagaser, Michael Bernard

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A COMPUTATIONAL COMPARISON OF THE PRIMAL SIMPLEX AND RELAXATION ALGORITHMS FOR SOLVING MINIMUM COST FLOW NETWORKS

by

Michael B. Sagaser

March 1989

Thesis Advisor: R. Kevin Wood

Approved for public release; distribution is unlimited

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (*If Applicable*): 36	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (*If Applicable*)	9 Procurement Instrument Identification Number
8c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title (*Include Security Classification*): A Computational Comparison of the Primal Simplex and Relaxation Algorithms for Solving Minimum Cost Flow Networks

12 Personal Author(s): Michael B. Sagaser

13a Type of Report	13b Time Covered	14 Date of Report (*year, month,day*)	15 Page Count
Master's Thesis	From To	March 1989	81

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Networks, Minimum Cost Flow Problems, Primal Simplex, Relaxation Method, Optimization, Lagrangian Relaxation.

19 Abstract (*continue on reverse if necessary and identify by block number*)

This thesis examines the relative computational efficiencies of two advanced network minimum cost flow problem solution methodologies: the primal simplex specialization to networks developed by Bradley, Brown and Graves (1977)--GNET and XNET, and a Lagrangian relaxation method developed by Berstekas and Tseng (1988)--RELAX-II and RELAXT-II. Additionally, the relaxation method description is clarified and potential implementation improvements are investigated.

Research by Bertsekas and Tseng has shown the relaxation codes to be on the order of four to five times faster than the primal simplex codes. This thesis fails to duplicate those results. While the relaxation codes do perform faster in many circumstances when solving purely random problems, the primal simplex codes are still closely competitive. In particular, the primal simplex codes appear more efficient at solving capacitated transshipment problems in networks with an echelon structure, and in networks with many more sinks than sources. Primal simplex codes also require about half the computer storage space of the relaxation codes.

The research has produced compelling evidence that the relaxation algorithms can be further refined. All indications appear to reinforce the desirability of prioritizing by absolute deficit the node selection process used in both relaxation codes. Further research is recommended.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified
22a Name of Responsible Individual: Kevin Wood	22b Telephone (*Include Area code*): (408) 646-2523; 22c Office Symbol: 55wd

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted

All other editions are obsolete

security classification of this page: Unclassified

Approved for public release; distribution is unlimited.

A Computational Comparison of the
Primal Simplex and Relaxation Algorithms
for Solving Minimum Cost Flow Networks

by

Michael Bernard Sagaser
Captain, United States Marine Corps
B.S., University of Arizona 1978

Submitted in partial fufillment of the requirements for
the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1989

ABSTRACT

This thesis examines the relative computational efficiencies of two advanced network minimum cost flow problem solution methodologies: the primal simplex specialization to networks developed by Bradley, Brown and Graves (1977)--GNET and XNET, and a Lagrangian relaxation method developed by Bertsekas and Tseng (1988)--RELAX-II and RELAXT-II. Additionally, the relaxation method description is clarified and potential implementation improvements are investigated.

Research by Bertsekas and Tseng has shown the relaxation codes to be on the order of four to five times faster than the primal simplex codes. This thesis fails to duplicate those results. While the relaxation codes do perform faster in many circumstances when solving purely random problems, the primal simplex codes are still closely competitive. In particular, the primal simplex codes appear more efficient at solving capacitated transshipment problems in networks with an echelon structure, and in networks with many more sinks than sources. Primal simplex codes also require about half the computer storage space of the relaxation codes.

The research has produced compelling evidence that the relaxation algorithms can be further refined. All indications appear to reinforce the desirability of prioritizing by absolute deficit the node selection process used in both relaxation codes. Further research is recommended.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

A frequently solved problem in the mathematical programming community today is the *minimum cost flow problem* (MCFP) on a network. That this is so reflects both the intuitive appeal of representing certain practical problems in terms of a network of arcs and nodes, and the fact that minimum cost flow problem solution algorithms have advanced to the point where enormous problems can be solved efficiently.

Many very practical situations can be represented in terms of flows through a system of arcs and nodes: products through a distribution network, water or petrochemicals through a pipe network, traffic through a road network, etc. Several quantitative fields depict phenomena in terms of a network flow model, including the U. S. military. Personnel assignment, ammunition distribution, optimal pack-out designs, inventory management , scheduling and planning are just some of the military uses of the network flow model (Rapp 1987, Staniec 1984 and Yorio 1988). Consequently, many military professionals have an interest in the effective formulation and the efficient solution of minimum cost flow problems. This thesis addresses the ability of modern MCFP solution algorithms to solve large scale problems by comparing two highly regarded approaches to solving the MCFP: the *primal simplex* and a newly introduced method known as the *relaxation method.*

The MCFP is based on a network which is a directed graph with a set of nodes N and a set of arcs A, each arc directed from its tail node to its head node, identified by a subscript α. Some nodes may have exogenous supply (a *source* node) or exogenous demand (a *sink* node). Nodes with neither exogenous supply or demand are pure transshipment

nodes. Associated with each node is a flow balance constraint which states that the flow of arcs into the node, including any exogenous supply, must equal the flow of arcs out of the node, including any exogenous demand. Each arc has associated with it a linear cost per unit flow c_α, and upper and lower bounds to the flow allowed through the arc, u_α and l_α respectively. The goal of the minimum cost network flow model is to determine a flow scheme that minimizes the total costs associated with shipping a specified product through the arcs of the network while ensuring that all node demands and arc flow limitations are met. If the flow passing through arc α is x_α, then the precise statement of the problem is:

$$\text{Minimize} \quad \sum_{k \in A} c_\alpha x_\alpha \tag{1.1}$$

$$\text{Subject to} \quad \sum_{\alpha \in A \text{ with tail } i} x_\alpha - \sum_{\alpha \in A \text{ with head } i} x_\alpha = b_i, \quad i \in N \tag{1.2}$$

$$l_\alpha \leq x_\alpha \leq u_\alpha, \quad \alpha \in A \tag{1.3}$$

where b_i is the exogenous supply of node i.

The MCFP can be solved as a general linear program with a constraint for each node and a variable for each arc. There are, however, far more efficient specializations of general linear programming algorithms that take advantage of the special structure of network problems. It is these specialized network algorithms that have permitted the mathematical programers of today to solve very large scale military and commercial problems efficiently.

The transportation problem, proposed by Hitchcock (1941), is the first instance of a MCFP to become widely known. Hitchcock presented a solution process that closely resembles the primal simplex methodology. Dantzig (1951) showed that the transportation

problem is an instance of a linear program and that it can be solved by his simplex algorithm, and in fact developed a special variant of the simplex algorithm to solve transportation problems. Orden (1956) showed that the more general transshipment MCFP can be solved by these same methodologies. Several approaches to solving the MCFP that are not primal simplex were subsequently proposed: the out-of-kilter algorithm by Fulkerson (1961); the primal-dual by Ford and Fulkerson (1957); the dual by Balas and Hammer (1962). Several investigators continued to pursue the primal simplex method and developed efficient codes for solving large scale MCFPs. See Mulvey (1974), Harris (1976) and Langley, Kennington and Shetty (1974). (Bradley, Brown and Graves 1977)

By the late 1970s the most efficient algorithm for solving network flow problems was widely accepted to be the primal simplex specialization as exemplified by Bradley, Brown and Graves (1977). This primal simplex solution algorithm for networks was implemented in an efficient Fortran code called GNET; it is described at length by its authors. Several variations of the basic GNET implementation are also investigated by Bradley *et al.*, including a code called XNET, which specializes to networks with relatively many sinks compared to sources, and is known as the *aggregated successors* version of GNET.

A new algorithm was introduced by Bertsekas and Tseng (1988) which does not belong to any previous category of network solution algorithms. This new method essentially applies what are generally considered to be nonlinear programming techniques to the dual of the network, a dual based on a Lagrangian relaxation of the MCFP, hence the name *relaxation method.* The implementation of the relaxation methodology exists today as a pair of Fortran codes, available from Bertsekas and Tseng, called RELAX-II and RELAXT-II. These two codes are reported by their authors to be between four and five times faster at solving randomly generated minimum cost flow problems than a primal simplex code written by Grigoriadis and Hsu (1988) called RNET.

B. PURPOSE

This thesis primarily investigates the relative efficiencies of the primal simplex network codes, GNET and XNET, and the newer relaxation codes RELAX-II and RELAXT-II. Two measures of effectiveness will be considered: the amount of computer running time needed to attain an optimal solution and the total computer storage needed to implement the procedure.

Additional goals of this thesis are to generalize and clarify the description of the relaxation methodology algorithms, and to study the particular algorithmic implementations to determine whether improvements can be made.

The primal simplex code versions evaluated are the original, unmodified GNET/Depth and XNET as presented by Bradley, Brown and Graves in 1977. The relaxation codes evaluated are versions 2.1 of RELAX-II and RELAXT-II, as introduced in 1986.

C. METHOD

There is no widely accepted testing method to compare the relative merits of network solvers. The most often used technique is to generate a series of artificial test problems and then base performance decisions on the resulting solution times. One drawback to this approach is that the test problems that can be generated do not often share the same structural characteristics of "real-world" problems since they must usually be created randomly. Some codes, like GNET and XNET, are advertised to take advantage of the structure of problems formulated from real applications. Is it possible to generate problems that contain convincing real-world structure, or should a set of widely accepted real test problems be gathered? This question will be addressed, but not answered completely.

To produce test problems for this thesis, a network problem generator called NETGEN, developed by Klingman, Napier and Stutz (1974), is used to generate a set of forty standard network problems which include transportation, assignment and capacitated

and uncapacitated transshipment networks. The NETGEN standard problems are used as a set of workable test problems by some, and are in fact the basis for the computational comparisons performed by Bertsekas and Tseng. They are used here to facilitate a comparison of results.

Utilizing NETGEN is not an optimal approach to creating test problems. NETGEN randomly generates the distribution of supply and demand nodes, the arc costs, and the placement of arcs within the network. Real-world networks are often constructed over a particular geographical area, e.g., a series of ports receiving some supply which must be shipped to warehouses inland which in turn get transshipped to demand points further inland. Special relationships often exist between flow costs and the topology of the network, and flow rates may be limited by geographic constraints. In short, a purely random structure does not exist in real life. However, because of its wide distribution and familiarity to most mathematical programmers, NETGEN has been the usual tool used to test new solution algorithms.

Another less well known network test problem generator developed by Bonwit (1984) is used in this study. Called VSNET, this generator takes into account some of the general structure characteristics often visible in real-world problems, particularly the geographical echelon characteristic discussed above. Through extensive testing, Bonwit established that GNET consistently solved VSNET problems faster that NETGEN problems of comparable size, indicating a dependence of an algorithm's practical efficiency on network structure. The version of VSNET in use here does not produce assignment problems, however, only transportation and transshipment problems.

This thesis uses both the NETGEN and VSNET problem generators to create test networks. NETGEN is chosen so that comparisons can be made with the computational experiments made by Bertsekas and Tseng. VSNET is chosen so that the effects of a

different problem structure can be observed. No real world problems are investigated because of the difficulty of reproducibility and acceptance among the wider mathematical programming community. This is not considered the best solution to the testing dilemma, but it is the only reasonable approach that could be made in view of the current state of algorithm testing technology.

D. OVERVIEW

Chapter II derives in detail the basic theory behind the relaxation methodology and presents the basic relaxation algorithm. In Chapter III the results of the computational comparison experiments are reported. Chapter IV suggests approaches to improving the relaxation method implementation by means of several data sorting schemes and other modifications to the implementing code. Conclusions are presented in Chapter V.

II. THE RELAXATION ALGORITHM

This chapter develops the relaxation methodology for solving minimum cost flow network problems. The development generally follows that of Bertsekas and Tseng (1988), but concentrates only on the ordinary network flow problem, excluding the network with gains. All vector quantities are in bold type.

The method essentially operates by ascending along a dual function based on the Lagrangian relaxation of the problem. In the past, Lagrangian relaxation has been widely used to solve large integer programming problems, where one can often observe a relatively simple problem complicated by a set of side constraints that can be partitioned out of the total set of constraints and placed in the objective function with some associated penalty cost (Fisher 1985). To implement this idea in network flow problems, all flow balance equations (1.2) are placed in the objective function in the relaxation methods. One can adapt what are normally considered nonlinear programming techniques, iteratively computing directional derivatives, to discover favorable directions of improvement in the dual. By further enforcing complementary slackness with the primal solution, an optimal feasible solution will ultimately be obtained. This is the essential characteristic of the method which will now be developed.

A. THE MINIMUM COST FLOW PROBLEM

The minimum cost flow problem (MCFP) described in Chapter I will be reiterated here in a form more suitable for deriving the relaxation algorithm.

The MCFP on a network is based on a directed graph consisting of a set of nodes N and a set of arcs A, each arc being identified by the ordered pair of nodes (i,j). For simplicity, the development of the relaxation methodology in this chapter will assume that

only one arc connects any two nodes, although the computer implementation of the method allows for multiple arcs. For each arc (i,j) there exists a flow x_{ij} and an associated cost per unit flow c_{ij}. Let l_{ij} and u_{ij} represent the lower and upper bounds on the flow of arc (i,j), respectively. (Occasionally, the notation $\alpha = (i,j)$ will be used to depict an arc for simplicity, e.g., x_α, u_α and l_α.) The basic MCFP problem is stated as

$$\text{Minimize} \qquad z = \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.1}$$

$$\text{Subject to} \qquad \sum_{m|(m,i)\in A} x_{mi} - \sum_{m|(i,m)\in A} x_{im} = -b_i \quad \forall\, i \in N \tag{2.2}$$

$$l_{ij} \le x_{ij} \le u_{ij} \quad \forall\, (i,j) \in A\,, \tag{2.3}$$

which has optimal solution $\mathbf{x}^*$ and optimal objective function value z^*.

The above problem statement differs from that of Bertsekas and Tseng in that the right hand side of (2.2) has been explicitly included and not required to be to zero. This generalization is done to more closely align the statement of the relaxation method to the primal simplex method for those readers already familiar with the latter. Equation (2.2) is written as the negative of equation (1.2) so that the theoretical model developed here agrees with the actual Fortran implementation of RELAX-II and RELAXT-II.

A Lagrangian function $L(\mathbf{x},\mathbf{p})$ is created by relaxing the flow balance constraints (2.2) and placing them in the objective function, with an associated penalty for violation of the constraints. The penalty term is p_i, called the *price* of node i. This new function is

$$L(\mathbf{x},\mathbf{p}) = \sum_{(i,j)\in A} c_{ij} x_{ij} + \sum_{i\in N} p_i \Big(\sum_{m|(m,i)\in A} x_{mi} - \sum_{m|(i,m)\in A} x_{im} + b_i \Big)$$

$$= \sum_{(i,j)\in A} (c_{ij} + p_j - p_i)\, x_{ij} + \sum_{i\in N} b_i p_i . \qquad (2.4)$$

Let the Lagrangian dual function q(**p**) be defined as

$$q(\mathbf{p}) = \min_{l_{ij}\le x_{ij}\le u_{ij}} L(\mathbf{x}, \mathbf{p}). \qquad (2.5)$$

The Lagrangian dual problem to the MCFP formulation given in equations (2.1) - (2.3) is then to maximize q(**p**), subject to no constraints on **p**. If **p*** is the value of the **p** vector that optimizes q(**p**) then q(**p***)=z*, although an optimal **x** for q(**p***) may not be feasible for the primal MCFP as written in equations (2.1) - (2.3). To assure a direct correspondence between the Lagrangian dual and the linear programming dual to MCFP (and thus assure that an optimal **x** for q(**p***) is also primal-feasible) it is necessary to add an additional restriction to (2.5). Accordingly, define, for any price vector **p**, the arc (i,j) as being

inactive if	$c_{ij} + p_j - p_i > 0,$	(2.6)
balanced if	$c_{ij} + p_j - p_i = 0,$ and	(2.7)
active if	$c_{ij} + p_j - p_i < 0.$	(2.8)

Also define within the context of (2.5)

$x_{ij} = l_{ij}$	for inactive arcs,	(2.9)
$l_{ij} \le x_{ij} \le u_{ij}$	for balanced arcs, and	(2.10)
$x_{ij} = u_{ij}$	for active arcs.	(2.11)

Equations (2.9) - (2.11) together constitute the additional restriction necessary to assure the direct correspondence between the Lagrangian and the linear programming duals; they are the complementary slackness conditions for the MCFP. Fisher (1985) discusses more completely the relationship between the Lagrangian and linear programming duals, and Rockafellar (1984) also addresses this relationship.

It is useful to identify a scalar quantity that represents the difference between the flow into and out of node i, called the *deficit* of node i. This quantity, taking into account any supply or deficit (demand) already existing at the node, is

$$d_i = \sum_{m|(i,m)\in A} x_{im} - \sum_{m|(m,i)\in A} x_{mi} - b_i \quad \forall\, i \in N .$$

The relaxation method adopts what is essentially a nonlinear programming strategy to solve linear network problems. It does this by operating on the Lagrangian dual (2.5), attempting to find a price vector direction of change that will improve the value of q(**p**) by successively calculating a directional derivative and adjusting the vector **p**. If the opportunity arises a *flow augmentation*, defined in the next paragraph, is performed to reduce primal infeasibility. Since the algorithm always operates on the dual of the network, dual feasibility is maintained. Once a favorable direction has been found, changes to the price vector **p** and to the flow vector **x** are accomplished in such a way that complementary slackness (equations (2.9) - (2.11)) is always maintained.

Given a vector pair (**x**,**p**) satisfying complementary slackness, a sequence of nodes $\{n_1, n_2, ..., n_k\}$ is a *flow-augmenting path* if the deficit of n_1 is strictly negative, the deficit of n_k is strictly positive, and for m=1, 2, ..., k-1, either there exists a balanced arc $\alpha = (n_m, n_{m+1})$ with $x_\alpha < u_\alpha$, or there exists a balanced arc $\alpha' = (n_{m+1}, n_m)$ with $x_{\alpha'} > l_{\alpha'}$.

Furthermore, if P^+ is the set of nodes in a path directed from n_1 towards n_k, and P^- is the set of nodes in a path directed from n_k towards n_1, the *capacity* of the path will be

$$\upsilon = \min\{ d_{nk}, -d_{n1}, \{ (u_\alpha - x_\alpha) \mid \alpha \text{ in } P^+\}, \{(x_{\alpha'} - l_{\alpha'}) \mid \alpha' \text{ in } P^- \}.$$

Flow augmentation consists of forcing an additional amount of flow υ along a path that starts at a node with negative deficit (surplus) and ends at a node with positive deficit. In this way the absolute deficit of the two extreme nodes on a flow-augmenting path will be reduced, while the deficits of the intervening nodes will be unaffected.

The process of solving MCFP with the relaxation method begins by setting all flows in the network to zero (unless the user provides initial flow and price vectors that satisfies complementary slackness in an attempt to accelerate the solution process) so that the initial deficit for each node is simply its demand (positive deficit) or supply (negative deficit), as required by the deficit equation. Define $\mathbf{e}_i$ to be the i^{th} unit vector associated with increasing the value of p_i, while all other components of $\mathbf{p}$ remain constant. Also define an initially empty set S that contains all nodes being considered for a price change.

For the price vector existing at the beginning of each relaxation iteration, a node i with positive deficit is selected and placed in S. It is determined whether the dual function (2.5) can be improved by altering the price of node i by taking the directional derivative of the dual function in the $-\mathbf{e}_i$ direction, at the current price vector. Why the decreasing price direction is appropriate for a node with positive deficit will be addressed in Section D of this chapter. If the dual function cannot be improved by decreasing p_i the algorithm then looks along balanced arcs for a node adjacent to S with a negative deficit. If such a node is found, flow can be "pushed" from the negative deficit node to node i, thus reducing the total absolute deficit of both node i and of the node that is found to have a negative deficit.

If a price adjustment is unfavorable, and there is no adjoining node with negative deficit, S is expanded by the addition of a node incident to node i, say node i'. Now it is determined whether the dual function can be improved by a simultaneous reduction of both p_i and $p_{i'}$ by taking the directional derivative of the dual function, evaluated at the current price vector, in the $-(\mathbf{e}_i + \mathbf{e}_{i'})$ direction. Again, if the price reduction turns out to be unfavorable, an attempt is made to find a node adjacent to S with negative deficit. If no flow can be pushed, another adjacent node is added to S, and so on. In practice, and by purposeful design, most price changes occur when S contains only one node (along a single coordinate direction) because a single node price adjustment is more computationally efficient than a multiple node price adjustment.

The above iterative procedure will necessarily end when either the price vector has been adjusted or a flow has been pushed from some node with negative deficit to the starting node i, as demonstrated by the theorem below. The algorithm itself will terminate when **x** satisfies primal feasibility, i.e., the deficit of each node equals zero. Note that there is a parallel case in which a node with negative deficit is initially selected for membership in S. When this is attempted, the process remains the same as outlined above with the exception that one now looks for a price increase for the set of nodes in S, or a node with positive deficit to push flow to.

Theorem: Given a flow and price vector (**x**,**p**), satisfying (2.9) - (2.11), and given that there exists at least one node with non-zero deficit, then it is possible to perform either a price adjustment or a flow augmentation on the network.

Proof: This proof is essentially that given by Bertsekas (1985) and is illustrative of the relaxation methodology.

Define the set S of scanned nodes to which price adjustments are to apply, and a set L of labeled nodes. After making both sets empty follow this procedure:

Step 1. Begin by picking some node i with positive deficit and placing it in S. (There is a parallel, symmetric negative deficit case that is not treated here.)

Step 2. Create a set L consisting of all nodes $m \notin S$ such that there exists an arc (m,k) directed into S which is balanced and has $x_{mk} < u_{mk}$, or there exists an arc (k,m) directed out of S which is balanced and has $x_{km} > l_{km}$.

Step 3. Select some node in L to bring into S. If a node in L with negative deficit has been found, stop. If a point is reached where all nodes in the network are either in S or all nodes in L have nonnegative deficit, stop. Otherwise, go to step 2.

There are two possible terminations to this process. The first is that a node with negative deficit is found, in which case a flow-augmenting path has been found and the total network deficit $\Sigma|d_i|$ can be reduced by 2υ (twice the capacity of the flow-augmenting path). The second possibility is that every node in L has a nonnegative deficit. Let L' be the complement of L. Then L' must be nonempty since $\Sigma_{i \in S} d_i > 0$, but $\Sigma_{i \in N} d_i = 0$. Therefore, there must exist either an arc (k,m) with $k \in L$ and $m \in L'$ that is active (flow at u_{mk}), or there must exist an arc (m,k) with $k \in L$ and $m \in L'$ that is inactive (flow at l_{km}). Let δ be a scalar defined as

$$\delta = \min \{\{ -(c_{km} + p_m - p_k) | (k,m) \text{ active } \},$$

$$\{(c_{km} + p_m - p_k) | (k,m) \text{ inactive } \}\}. \qquad (2.12)$$

Set $p_i = p_i - \delta$ for all nodes $i \in S$. Since the $(\mathbf{x},\mathbf{p}')$ is still an integer vector pair satisfying complementary slackness, changing $(\mathbf{x},\mathbf{p})$ to $(\mathbf{x},\mathbf{p}')$ is in fact carrying out a valid price adjustment. QED.

B. THE DUAL ASCENT STEP

The goal of the dual ascent step in the relaxation algorithm is to improve (increase) the value of the dual function (2.5) by adjusting the price of a selected set of nodes, or in some cases, a single node. As indicated, the algorithm begins by selecting a node with deficit, say positive, and determining whether $q(\mathbf{p})$ can be improved by reducing the price of the selected node. Specifically, the directional derivative of (2.5) is calculated in the direction of decreasing price for the selected node(s). If the derivative (evaluated at the current price vector) is favorable, then it is advantageous to decrease the price of the selected node(s). The negative deficit case is analogous and will be treated separately. The actual computation of the directional derivative is done incrementally in the implementation of the relaxation algorithm by means of an identity derived below.

Given that a price change has been found to be favorable for some set of nodes S, the step size δ in (2.12) corresponds to the first break point of the piecewise linear dual function along some ascent direction. The first break point reached in this manner may or may not be located at the maximum value of the dual function along the direction implied by the nodes currently in S. Bertsekas and Tseng report that it is possible to find an optimal price adjustment stepsize that maximizes the value of the dual function in the chosen ascent direction. The technique for doing this is quite simple and involves testing the sign of the directional derivative of the dual function at successive break points along the ascent direction. If the sign continues to indicate that more can be gained by further price change, then the price is adjusted accordingly and the directional derivative is again evaluated. This process is called a *line search* and is in fact implemented in the codes of both RELAX-II and RELAXT-II.

The particular case in which a node s with positive (negative) deficit comprises S and the relaxation algorithm immediately finds a favorable directional derivative, before any additional nodes are added to S, is called a *single node iteration.* In this case p_s, $S=\{s\}$

will be decreased (increased) by δ, perhaps repeatedly via a line search, and the iteration will terminate. The only price to change will have been that of node s. The associated change in flow will reduce the absolute value of the deficit of node s at the expense of possibly increasing the absolute of the deficit of neighboring nodes.

1. The Decreasing Price Directional Derivative

The specific problem here is to determine the directional derivative of (2.5) in the decreasing price direction for the selected nodes, given that the initially selected node has a positive deficit. In this case it is expected that the price of any nodes in S will be reduced if the slope of the dual function at the current price vector is sloped negatively along the coordinate corresponding to a reduction in price for nodes in S, thus improving the value of the dual function. Recall that the directional derivative will have to be calculated once each time another node is added to S, the set under consideration for a price reduction. The general expression for a directional derivative in the direction of a vector **w** is

$$C^{\mathbf{w}}(\mathbf{p}) = \min_{t \to 0^+} \sum_{(i,j) \in A} \frac{L(\mathbf{x}, \mathbf{p} + t\mathbf{w}) - L(\mathbf{x}, \mathbf{p})}{t} \qquad (2.13)$$

The direction of initial interest is $w_i = -1$ for $i \in S$ and $w_i = 0$ for $i \notin S$, where S is a connected set of nodes, all of which have nonnegative deficit. This directional derivative will be denoted $C^-_S(\mathbf{p})$. The following paragraphs develop an expression which is used to compute this directional derivative in the relaxation codes.

In evaluating $C^-_S(\mathbf{p})$, we note that there will be $2(|N|+|A|)$ terms in numerator of (2.13). All those terms associated with arcs between pairs of nodes in S, or between pairs of nodes not in S will cancel, as will all the terms associated with nodes not in S. Thus, only those terms associated with arcs crossing the boundary of S and those terms associated with nodes in S need to be considered. The boundary arcs fall into one of six

categories: either they are incident *into* S and active, balanced or inactive, or the are incident *out of* S and active, balanced or inactive. For each case listed above the price of the nodes in S will be adjusted by t as per (2.13) and the resulting expression evaluated.

First consider any arcs (i,j) inbound to S. Since we wish to test the favorableness of reducing price, reduce the price of node j by t. Referring to (2.5), if the arc is inactive then $c_{ij} + p_j - p_i$ is positive and, since $L(\mathbf{x},\mathbf{p})$ is to be minimized, the flow on arc (i,j) must be at its lower bound. Accordingly, for t sufficiently small

$$[(c_{ij} + (p_j - t) - p_i) l_{ij} - (c_{ij} + p_j - p_i) l_{ij}]/t = -l_{ij}. \qquad (2.14)$$

Likewise, for an active arc (i,j), $c_{ij} + p_j - p_i$ is negative, so the flow on (i,j) must be at its upper bound to minimize $L(\mathbf{x},\mathbf{p})$, yielding

$$[(c_{ij} + (p_j - t) - p_i) u_{ij} - (c_{ij} + p_j - p_i) u_{ij}]/t = -u_{ij}. \qquad (2.15)$$

If arc (i,j) is balanced then $c_{ij} + p_j - p_i = 0$, but reducing p_j by any amount will drive the quantity negative. In this case the flow of (i,j) must be set to its upper bound, or

$$[(c_{ij} + (p_j - t) - p_i) u_{ij} - (c_{ij} + p_j - p_i) x_{ij}]/t = -u_{ij}. \qquad (2.16)$$

Now consider any arcs (i,j) that are outbound from S. Reduce the price of node i by t. Again, by referring to (2.5) it can be seen that for inactive arcs with $c_{ij} + p_j - p_i$ positive, the flow on (i,j) is at its lower bound, which means that

$$[(c_{ij} + (p_j - t) - p_i) l_{ij} - (c_{ij} + p_j - p_i) l_{ij}]/t = -l_{ij}. \qquad (2.17)$$

For any active arc (i,j), $c_{ij} + p_j - p_i$ is negative, requiring the flow on (i,j) to be at its upper bound, or

$$[(c_{ij} + p_j - (p_i - t))\, u_{ij} - (c_{ij} + p_j - p_i)u_{ij}]\,/t = u_{ij}. \tag{2.18}$$

If arc (i,j) is balanced, $c_{ij} + p_j - p_i = 0$, but decreasing p_i by any amount will drive it positive, meaning that the flow x_{ij} must be set to its lower bound, giving

$$[(c_{ij} + p_j - (p_i - t))\, l_{ij} - (c_{ij} + p_j - p_i)x_{ij}]\,/t = l_{ij}. \tag{2.19}$$

Finally, the terms of $C^{-}_{S}(\mathbf{p})$ associated with the nodes in S yield

$$\Big(\sum_{i \in S} b_i (p_i - t) - \sum_{i \in S} b_i p_i\Big)/t = -\sum_{i \in S} b_i. \tag{2.20}$$

Summing (2.14) - (2.20) gives an expression for the directional derivative of (2.5) in the decreasing price direction for the direction implied by the selected nodes in S is

$$\begin{aligned} C^{-}_{S}(\mathbf{p}) = & \sum_{\substack{i \in S, j \notin S \\ (i,j)\ \text{Active}}} u_{ij} + \sum_{\substack{i \in S, j \notin S \\ (i,j)\ \text{Balanced}}} l_{ij} + \sum_{\substack{i \in S, j \notin S \\ (i,j)\ \text{Inactive}}} l_{ij} \\ & - \sum_{\substack{i \notin S, j \in S \\ (i,j)\ \text{Active}}} u_{ij} - \sum_{\substack{i \notin S, j \in S \\ (i,j)\ \text{Balanced}}} u_{ij} - \sum_{\substack{i \notin S, j \in S \\ (i,j)\ \text{Inactive}}} l_{ij} - \sum_{i \in S} b_i \end{aligned} \tag{2.21}$$

which can be further simplified by adding and subtracting the term

$$\sum_{\substack{i \notin S, j \in S \\ (i,j)\ \text{Balanced}}} x_{ij} - \sum_{\substack{i \in S, j \notin S \\ (i,j)\ \text{Balanced}}} x_{ij}$$

to the right hand side of (2.21). After arranging terms, the final identity for the directional derivative is

$$C_S^-(\mathbf{p}) = d_S - \sum_{\substack{i \in S, j \notin S \\ (i,j)\ \text{Balanced}}} (x_{ij} - l_{ij}) - \sum_{\substack{i \notin S, j \in S \\ (i,j)\ \text{Balanced}}} (u_{ij} - x_{ij}), \tag{2.22}$$

where

$$d_S = \sum_{i \notin S, j \in S} x_{ij} - \sum_{i \in S, j \notin S} x_{ij} - \sum_{i \in S} b_i$$

is the total deficit of S.

2. The Increasing Price Directional Derivative

The same approach is used for the increasing price derivative as is used to develop (2.22). Here the initially selected node to enter S has a negative deficit and it is desired to determine whether (2.5) can be improved by increasing the price of the nodes in S, i.e., find out if the slope of the dual function at the selected price vector is positive. Again, for inbound arcs (i,j), increasing the price of node j by t (and using the same arguments) gives

$$[(c_{ij} + p_j - (p_i + t))u_{ij} - (c_{ij} + p_j - p_i)u_{ij}]/t = -u_{ij} \tag{2.23}$$

if (i,j) is active,

$$[(c_{ij} + p_j - (p_i + t))u_{ij} - (c_{ij} + p_j - p_i)x_{ij}]/t = -u_{ij} \tag{2.24}$$

if (i,j) is balanced,

$$[(c_{ij} + p_j - (p_i + t))l_{ij} - (c_{ij} + p_j - p_i)l_{ij}]/t = -l_{ij} \tag{2.25}$$

if (i,j) is inactive, while increasing the price of node i for outbound arcs (i,j) gives

$$[(c_{ij} + (p_j + t) - p_i)u_{ij} - (c_{ij} + p_j - p_i)u_{ij}]/t = u_{ij} \tag{2.26}$$

for (i,j) active,

$$[(c_{ij} + (p_j + t) - p_i)l_{ij} - (c_{ij} + p_j - p_i)x_{ij}]/t = l_{ij} \tag{2.27}$$

for (i,j) balanced,

$$[(c_{ij} + (p_j + t) - p_i)l_{ij} - (c_{ij} + p_j - p_i)l_{ij}]/t = l_{ij} \tag{2.28}$$

for (i,j) inactive. As before, the terms in S yield

$$(\sum_{i \in S} b_i(p_i + t) - \sum_{i \in S} b_i p_i)/t = \sum_{i \in S} b_i . \tag{2.29}$$

Summing (2.23) - (2.29) gives an expression for the directional derivative of (2.5) in the decreasing price direction for the nodes in S,

$$C_S^+(\mathbf{p}) = \underset{\substack{i\notin S, j\in S \\ (i,j)\ \text{Active}}}{\sum} u_{ij} + \underset{\substack{i\notin S, j\in S \\ (i,j)\ \text{Balanced}}}{\sum} l_{ij} + \underset{\substack{i\notin S, j\in S \\ (i,j)\ \text{Inactive}}}{\sum} l_{ij} + \sum_{i\in S} b_i$$

$$- \underset{\substack{i\in S, j\notin S \\ (i,j)\ \text{Active}}}{\sum} u_{ij} - \underset{\substack{i\in S, j\notin S \\ (i,j)\ \text{Balanced}}}{\sum} u_{ij} - \underset{\substack{i\in S, j\notin S \\ (i,j)\ \text{Inactive}}}{\sum} l_{ij} \qquad (2.30)$$

which can be simplified by adding and subtracting

$$\underset{\substack{i\in S, j\notin S \\ (i,j)\text{Balanced}}}{\sum} x_{xj} - \underset{\substack{i\notin S, j\in S \\ (i,j)\text{Balanced}}}{\sum} x_{ij}$$

to the right hand side of (2.30) and rearranging terms which gives

$$C_S^+(\mathbf{p}) = \underset{\substack{i\in S, j\notin S \\ (i,j)\ \text{Balanced}}}{\sum} (x_{ij} - u_{ij}) - \underset{\substack{i\notin S, j\in S \\ (i,j)\ \text{Balanced}}}{\sum} (l_{ij} - x_{ij}) - d_S, \qquad (2.31)$$

where d_S is the total deficit of S.

C. THE BASIC RELAXATION ALGORITHM

Each relaxation iteration begins with a flow and price vector satisfying complementary slackness. If starting flow and price vectors have not been provided by the user, the algorithm sets them to zero. Each iteration will produce another flow and price vector also satisfying complementary slackness. The process will terminate when no node with a deficit can be found. The algorithm presented below is from Bertsekas and Tseng (1988)

and treats only the case where nodes with positive deficits are selected for inclusion in S. The parallel case for selecting nodes with negative deficits is similar.

Step 1. Chose a node s, with with a positive deficit d_s. If there are none, terminate the algorithm. Set $S=\emptyset$ and $L=\{s\}$.

Step 2. Choose any $k \in L$ and let $S=S+\{k\}$, $L=L-\{k\}$.

Step 3. For each arc (k,m) directed out of S, if $x_{km} > l_{km}$ let $L=L+\{m\}$. For each arc (m,k) directed into S, if $x_{mk} < u_{mk}$ let $L=L+\{m\}$. Compute $C^-_S(\mathbf{p})$ and, if positive, go to step 5. If any node $m' \in L$ has negative deficit, go to step 4. Otherwise, go to step 2.

Step 4. Flow Augmentation: A flow augmenting path from m' to s has been found. Identify arcs directed from s to m' as belonging to set P^-, and arcs directed from m' to s as belonging to set P^+. Compute

$$\upsilon = \min\{ d_s, -d_{m'}, \{ (u_{kn} - x_{kn}) \mid (k,n) \in P^+\}, \{(x_{kn} - l_{kn}) \mid (k,n) \in P^- \}\}.$$

Let $x_{kn} = x_{kn} + \upsilon$, $\forall$ arcs $(k,n) \in P^+$, and let $x_{kn} = x_{kn} - \upsilon$ $\forall$ arcs $(k,n) \in P^-$. Go to step 1.

Step 5. Price Adjustment. Set

$$\delta = \min\{\{(p_k - p_m - c_{km}) \mid (k,m) \text{ is outbound from S and active}\},$$
$$\{(c_{mk} - p_m + p_k) \mid (m,k) \text{ is inbound to S and inactive}\}\}$$

and set

$x_{km} = l_{km}$, for all balanced arcs (m,k) outbound from S,

$x_{mk} = u_{mk}$, for all balanced arcs (m,k) inbound to S.

Set $p_k = p_k - \delta, \forall\ k \in S$. Go to step 1.

The relaxation iteration will terminate either when a flow augmentation (step 4) or a coordinate ascent (step 5) has occurred. The procedure is well defined since when one returns to step 2 from step 3 there is always one node in L that is not in S. When $S \neq \emptyset$ and $L = \emptyset$ it must be that there are no balanced arcs crossing the boundary of S. Thus it follows from (2.22) that

$$C_S^-(\mathbf{p}) = \sum_{k \in S} d_k > 0;$$

the procedure will therefore switch from step 3 to step 5 rather than switch to step 2 because an ascent direction has been found.

It is simple to show show that the relaxation procedure converges if the starting flow and price vectors are both integer. In this case δ is also an integer and the dual will be increased by an integer amount each time step 5 is performed. When a flow adjustment occurs in step 4 the dual cost does not change, and if the initial flow vector is integer then all subsequent flows will be integer since υ is always be integer. In view of these arguments, there can only be a finite number iterations between successive reductions in the dual cost so that the algorithm will terminate finitely with an optimal flow and price vector. If the starting flow and price vectors are not integer, the convergence analysis is far more complex and it is necessary to introduce some modifications to the basic relaxation methodology to assure convergence to near optimal solution. The essential elements of the proof are developed by Bertsekas and Tseng (1988) and Tseng (1986).

D. A NUMERICAL EXAMPLE

To illustrate how a typical dual ascent step would proceed, the following numerical example is offered. Suppose we have a five node, four arc network as shown in Figure 2.1(a). The costs and capacities are given in an edge list as follows:

Arc	Cost	Upper Bound
(1,i)	10	20
(i,2)	3	10
(3,i)	0	20
(i,4)	0	30

All node prices are assumed to begin at the values shown in Figure 2.1(a). To determine the flow levels of each arc at the beginning of this problem, apply definitions (2.6) - (2.11) as follows:

Arc (1,i) has $c_{1i} + p_i - p_1 = 10 + 25 - 5 = 30$: (1,i) is inactive, therefore $x_{1i} = 0$.
Arc (i,2) has $c_{i2} + p_2 - p_i = 3 + 10 - 25 = -12$: (i,2) is active, therefore $x_{i2} = 10$.
Arc (3,i) has $c_{i3} + p_i - p_1 = 0 + 25 - 15 = 10$: (3,i) is inactive, therefore $x_{3i} = 0$.
Arc (4,i) has $c_{i4} + p_4 - p_i = 0 + 20 - 25 = -5$: (4,i) is active, therefore $x_{4i} = 30$.

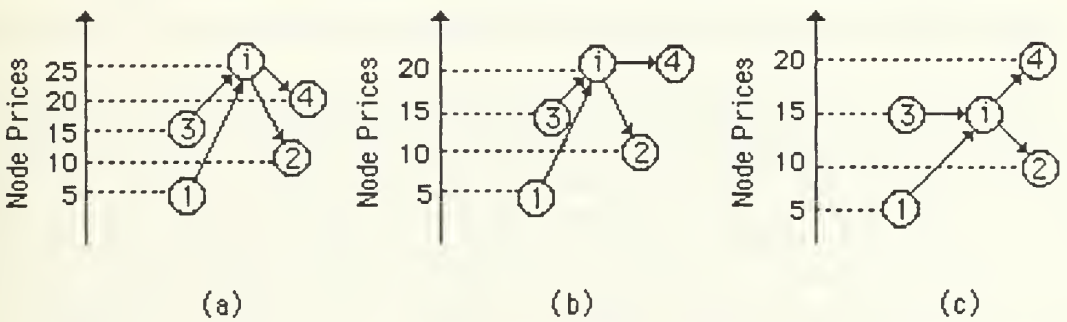

Figure 2.1. Illustration for the Numerical Example.

The flow situation is such that node i has net deficit of positive 40 (from the deficit equation), so we are now interested to discover whether the dual function can be improved

by means of a reduction in p_i. To determine this we turn to the expression for $C^-_S(\mathbf{p})$, equation (2.22). Since there are no balanced arcs at present, the directional derivative has a value of 40, indicating that p_i can be profitably reduced. Next, we need to decide just how far to reduce p_i to reach the first break point in the piecewise linear dual function. Applying step 5 of the relaxation algorithm, we see that the value of δ can be computed by

$$\delta = \min\{\{(p_k - p_m - c_{km}) \mid (k,m) \text{ is outbound from S and active}\},$$
$$\{(c_{mk} - p_m + p_k) \mid (m,k) \text{ is inbound to S and inactive}\}\}$$

which yields $\delta_1=\min\{30,12,10,5\}=5$, taking the arcs in the order in which flows are computed above. Finishing step 5 we reduce p_i by 5 to 20 and must now decide how to adjust the flows. To determine the flow status of the arcs we again apply definitions (2.6) - (2.11) as follows:

Arc (1,i) has $c_{1i} + p_i - p_1 = 10 + 20 - 5 = 25$: (1,i) is inactive, therefore $x_{1i} = 0$.
Arc (i,2) has $c_{i2} + p_2 - p_i = 2 + 10 - 20 = -7$: (i,2) is active, therefore $x_{i2} = 10$.
Arc (3,i) has $c_{3i} + p_i - p_3 = 0 + 20 - 10 = 5$: (3,i) is inactive, therefore $x_{3i} = 0$.
Arc (i,4) has $c_{i4} + p_4 - p_i = 0 + 20 - 20 = 0$: (i,4) is balanced.

Since arc (i,4) is now balanced, we complete step 5 by setting $x_{i4} = 0$. Note that the starting value of the dual function, obtained by using (2.4) and (2.5), can be computed as (30)0+(-12)10+(10)0+(-5)30 = -270 (point a in Figure 2.2) while the new value of the dual function, after applying the price change, is (25)0+(-7)10+(5)0+(0)0 = -70 (point b in Figure 2.2), indicating a net increase for the iteration.

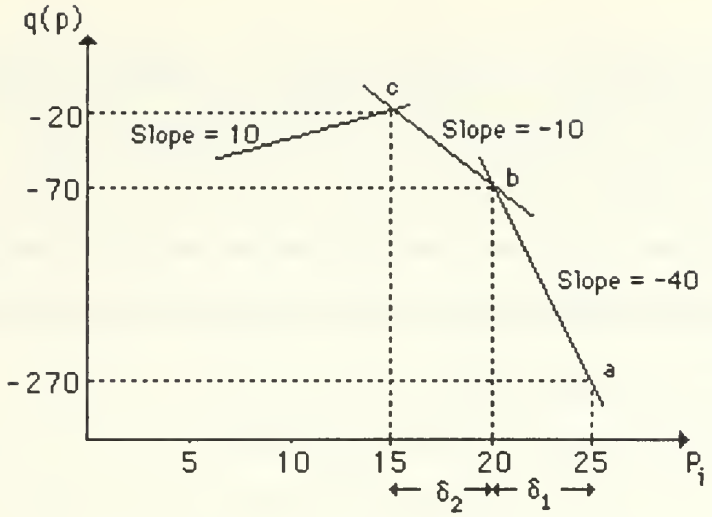

Figure 2.2. The Dual Function Surface.
The price of node i begins at a value of 25. Reduction by δ_1 increases the value of the dual function from a to b. At b a line search is instituted and a further decrease of p_i is found to be advantageous. After a second reduction of p_i of δ_2, the dual function is further improved by moving from b to c. At c $C^-_S(p)<0$, ending the relaxation iteration.

So far, a single node iteration has been successfully carried out since only the single price p_i has been reduced. To continue looking for favorable price reductions at this point constitutes the employment of the line search technique addressed earlier. To illustrate the line search we continue by determining whether a further reduction of p_i is favorable by computing $C^-_S(\mathbf{p})$ for the current flow and price vectors. Employing equation (2.22) we see that $C^-_S(\mathbf{p})=10$, indicating that a further reduction of p_i is warranted. The total allowable reduction is $\delta_2=\min\{25,7,5\}=5$, so that the value of p_i is lowered from 20 to 15. The new flow situation is:

Arc (1,i) has $c_{1i} + p_i - p_1 = 10 + 15 - 5 = 20$: (1,i) is inactive, therefore $x_{1i} = 0$.
Arc (i,2) has $c_{i2} + p_2 - p_i = 3 + 10 - 15 = -2$: (i,2) is active, therefore $x_{i2} = 10$.
Arc (3,i) has $c_{i3} + p_i - p_1 = 0 + 15 - 15 = 0$: (3,i) balanced.
Arc (i,4) has $c_{i4} + p_4 - p_i = 0 + 20 - 15 = 5$: (i,4) is inactive, therefore $x_{i4} = 0$.

The above flow picture has arc (3,i) balanced, we therefore complete the iteration by setting $x_{3i} = 20$, its upper bound. At the end of this second price adjustment we note that the dual function now has value (20)0+(-2)10+(0)20+(5)0 = -20 (point (c) in Figure 2.2). Finally, the directional derivative $C^{-}_{S}(\mathbf{p})$ now has value -10, since arc (3,i) is now providing 20 units of flow into node i while arc (i,2) is still at 10 units of flow out of node i. This terminates the relaxation iteration for node i.

III. COMPUTATIONAL COMPARISON OF PRIMAL SIMPLEX AND RELAXATION METHODOLOGIES

This chapter reports on the outcome of several computational efficiency comparisons between the relaxation and the primal simplex methodologies for solving minimum cost network flow problems. The relaxation methodology is represented by two implementations: version 2.1 of RELAX-II which is a straight implementation of the algorithm presented at the end of the previous chapter, and version 2.1 of RELAXT-II, which is different in that it maintains a separate dynamic data structure for all currently known balanced arcs. The primal simplex methodology is represented by the original version of GNET/Depth and XNET, a refinement of GNET/Depth that specializes to networks with relatively more sinks than sources, also known as the aggregated successors version of GNET; both are described by Bradley, Brown and Graves (1977).

A. DOCUMENTATION AND STORAGE REQUIREMENTS

Both relaxation codes are easily adapted from the VAX Fortran implementation that is available from Bertsekas and Tseng to VS Fortran on an IBM 3033AP computer. A minor translation chore of eliminating several DO WHILE loops and reducing a few variable names to be less than six characters in length is required because these two VAX Fortran features are not available in VS Fortran. Having done this, the user is required to write a small controlling program to read the network data, call the relaxation subroutines and produce the desired output files.

The documentation that is made available with the relaxation codes is adequate to allow a user to employ the codes to solve network problems in a straightforward fashion, but is not useful for understanding the functional details of the algorithms. Broad explanations

are provided in a theoretical framework, but one is left wondering about many implementation details. Consequently, it is necessary to puzzle out many important design features, such as node selection procedures, directional derivative calculations and multiple node iteration termination procedures. These and other items are important implementation aspects of the relaxation algorithms, a detailed description of which would greatly improve the employability of the algorithms.

Program storage requirements for RELAX-II are 18,516 bytes using 1129 source statements, when compiled under VS Fortran optimization level 3, while RELAXT-II requires 23,128 bytes and uses 1474 source statements. This compares unfavorably with GNET which requires 10,084 bytes of storage and uses 475 source statements and XNET which uses 11,025 bytes and 495 statements. Additionally, the dynamic storage requirements for both relaxation codes is considerably higher that for the primal simplex codes as can be seen in Table 3.1. These storage requirements differ from those reported by Bertsekas and Tseng (1988), who assert that RELAX-II uses 7.5 arc length and 7 node length arrays and that RELAXT-II uses 9.5 arc length and 9 node length arrays.

TABLE 3.1. MAJOR ARRAY SIZES

Code	Four Byte Integer Arrays		One Byte Logical
	Arc Length	Node Length	Node Length Arrays
RELAX-II	9	10	1
RELAXT-II	12	10	1
GNET	3	9	0
XNET	3	10	0

B. STANDARD NETGEN PROBLEMS

The forty standard NETGEN network problems developed by Klingman, Napier and Stutz (1974) are generated and run for each solver being evaluated: RELAX, RELAXT-II, GNET and XNET. The same parameters are used to generate problems for this thesis as

are used in the original NETGEN paper so that the test problems can be duplicated exactly by generating them as prescribed by the NETGEN authors. All solutions agreed with those obtained in the original NETGEN paper except for two: NETGEN-28 and NETGEN-29. The published solutions to these two problems are 122,582,531 and 105,050,119 respectively, while all solver codes in this investigation yield solutions of 122,582,559 and 105,050,170--a small difference that is not considered important to the overall test.

Only the actual solution processes for each algorithm are measured for run-time efficiency; data input time, data structure set-up time, and solution output time are not considered in the time measurements taken. All time measurements are obtained on an IBM 3033AP computer in time share using CMS version 5.0 and compiled by VS FORTRAN version 1.4.1 under optimization level 3.

Table A.1 contains the results of the standard NETGEN problem set tests. Table A.1 contains the results of the standard NETGEN problem set test. Figures 3.1 and 3.2 summarize the results. NETGEN 1-10 are 200 and 300 node transportation problems. NETGEN 11-15 are 400 node assignment problems. NETGEN 16-27 are 400 node capacitated transshipment problems broken down as follows: 16-19 have 20% of arcs capacitated, 20-23 have 40% of arcs capacitated, and 24-27 have 80% of arcs capacitated (see Table A.1 for a breakdown of the specific running times). NETGEN 28-35 are uncapacitated transshipment problems, the first four of 1000 nodes and the last four of 1500 nodes. NETGEN 36-40 are all large transshipment problems with the first three uncapacitated and the last two very slightly capacitated (with .7% of arcs capacitated). All have many more sinks than sources, and all contain both pure sources and sinks and transshipment sources and sinks of various numbers.

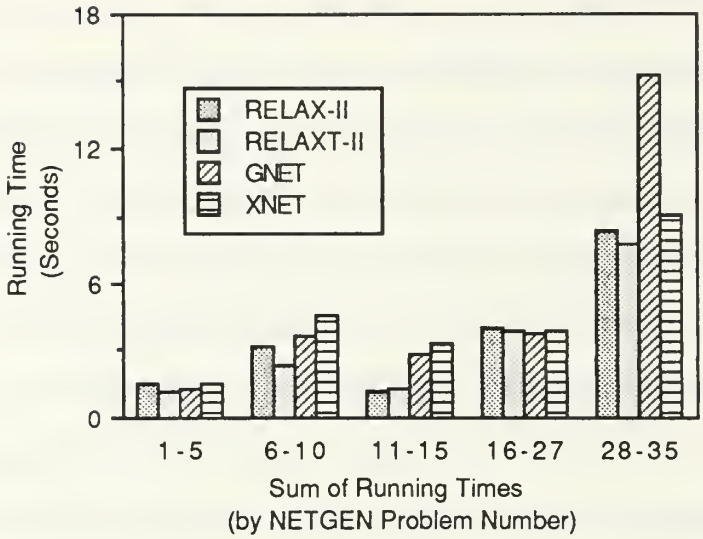

Figure 3.1. Running Times for the First 35 NETGEN Problems

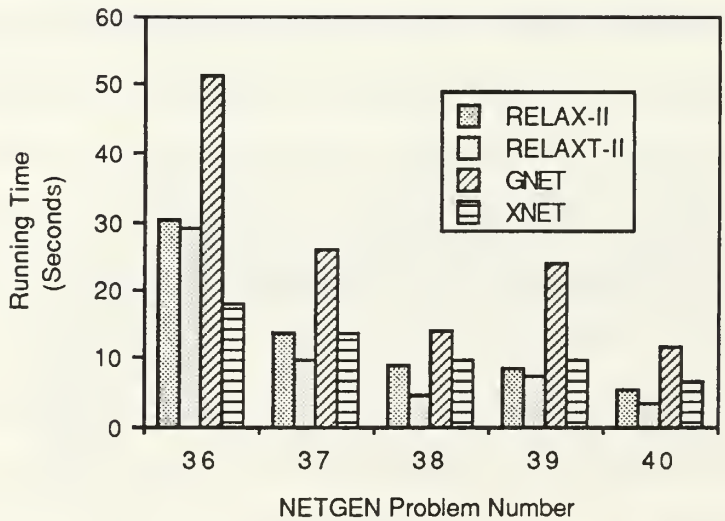

Figure 3.2. Running Times for NETGEN 36-40

It can be seen that the performance of the relaxation codes is slightly superior to the primal simplex codes for transportation (NETGEN 1-10) and assignment (NETGEN 11-15) problems, while they are clearly faster in uncapacitated transshipment (NETGEN 28-35) problems. Both primal simplex codes appear to be competitive when solving capacitated transshipment (NETGEN 16-27) problems.

It is interesting to point out that these results are far more favorable to the primal simplex codes than those conveyed by Bertsekas (1985), and Bertsekas and Tseng (1988), who reported a substantial superiority of the relaxation codes in transportation problems (a factor of three) and assignment problems (a factor of four). The RELAXT-II codes are superior to all other codes when solving problems in the NETGEN problem set, except that GNET runs are slightly better for lightly capacitated transshipment (NETGEN 16-27) problems. As might be expected, XNET is closely competitive on the large NETGEN problems shown in Figure 3.2, which all contain, to a greater extent than the other NETGEN problems, many more sinks than sources.

C. THE VSNET PROBLEM SET

Additional test problems are generated using a network generator called VSNET developed by Bonwit (1984). VSNET constructs a network as a series of echelons, with both the number of nodes in each echelon and the total number of echelons specified by the user, upon which a random set of arcs is placed. Six standard test problems are constructed for use throughout this thesis, three capacitated and three uncapacitated. Table 3.2 shows the parameters used to generate test problems using VSNET. As with all of the NETGEN problems, cost range is kept constant at between 1 and 100.

Table A.2 contains the results obtained from the VSNET problem set, and Figures 3.3 and 3.4 summarize these results.

TABLE 3.2. THE VSNET PROBLEM SET

VSNET#	Number of Nodes	Number of Arcs	Num. of Echelons	Total Supply	Num. of Sources	Number of Sinks
Capacitated Transshipmemnt Networks						
1	500	10000	3	100000	100	50
2	1000	20000	5	200000	125	75
3	5000	30000	6	1000000	400	400
Uncapacitated Transshipment Netwoks						
4	500	10000	3	100000	100	50
5	1000	18000	5	250000	100	100
6	5000	30000	6	1000000	400	400

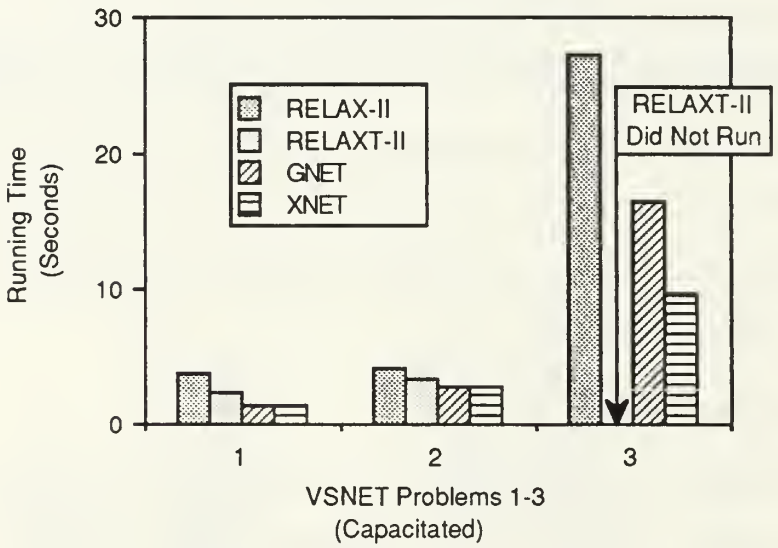

Figure 3.3. Capacitated VSNET Problems

The primal simplex codes can be seen to be relatively more efficient when solving VSNET capacitated transshipment networks (Figure 3.3). This is attributed to the fact that these networks are constructed with a structure that is found to be advantageous to the primal simplex codes by Bonwit; both GNET and XNET contain pricing heuristics that take advantage of the "real-world" structure that VSNET tries to duplicate. (Note that RELAXT-II does not run for VSNET-3--it produces a solution value of zero. A failure to

run satisfactorily turns out to be a recurring problem with RELAXT-II in several other test problems as well.)

For the uncapacitated transshipment VSNET problems (Figure 3.4), RELAXT-II is clearly most efficient, but GNET and XNET are closely competitive with RELAX-II. The relative improvement in the efficiencies of the primal simplex codes for these VSNET uncapacitated problems (as opposed to the NETGEN-generated problems) is most likely due to the structural differences of the two varieties of test networks. One sees the effect of the design features of GNET and XNET that take advantage of inherent structure.

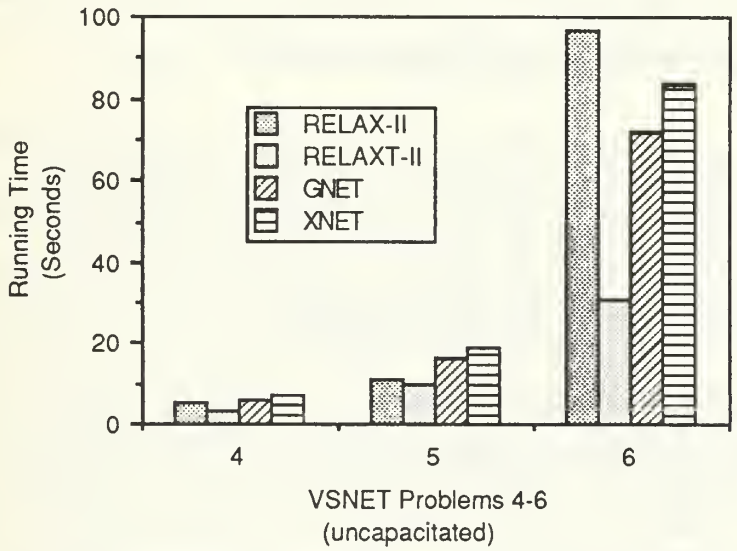

Figure 3.4. Uncapacitated VSNET Problems

Inspection of Table 3.2 reveals that VSNET 1-6 have more sources than sinks. This structure is considered by some to be unrealistic. The practical problems most often encountered by mathematical programmers in military and commercial problems tend to have many more sinks than sources and to expand as one moves into the echelon, e.g., a few production plants sending products to a few more warehouses which in turn send products to many more customers. This expanding echelon structure is fairly common in

practice and GNET and XNET are designed to take advantage of it. Note that both primal simplex codes do exhibit relative performance improvements even though the expanding echelon structure is not used in VSNET 1-6.

As an additional experiment, six more VSNET problems are generated that have all of the same basic parameters as VSNET 1-6, but with an expanding echelon structure imposed. Table 3.3 contains the running times for these problems which Figure 3.5 summarizes. Interestingly, the primal simplex codes are even more efficient relative to the relaxation codes than is the case in the original VSNET problem set, and in fact run competitively in two out of three uncapacitated transshipment networks. The inexorable deduction here is that structure is important to a solution algorithm.

TABLE 3.3. EXPANDING ECHELON VSNET PROBLEMS

VSNET# Equiv.	Num. of Sources	Num. of Sinks	RELAX-II	RELAXT-II	GNET	XNET
Capacitated Transshipment Networks						
1	10	390	3.16	1.95	0.81	0.76
2	10	465	9.48	DNR	2.38	2.01
3	5	1945	12.27	6.86	19.92	6.45
Uncapacitated Transshipment Networks						
4	10	390	9.03	7.49	4.02	5.02
5	10	465	13.51	DNR	13.20	15.86
6	5	1945	31.22	DNR	57.94	43.45

DNR: Did Not Run

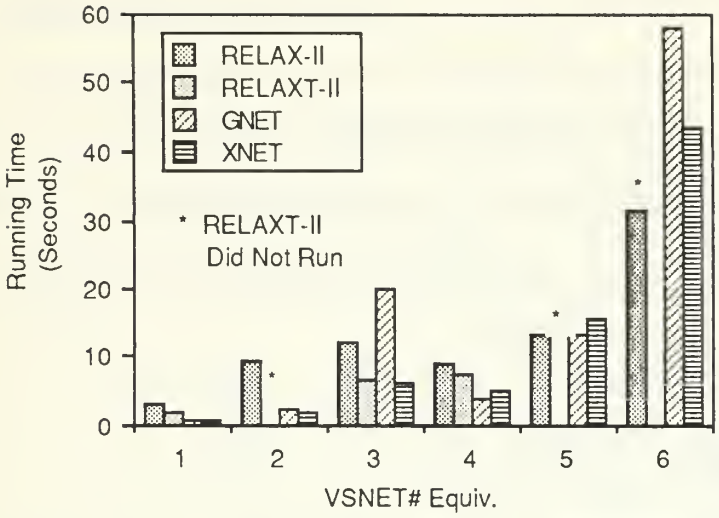

Figure 3.5. Expanding Echelon VSNET Problems

D. VARIATIONS OF THE NETGEN PROBLEMS

Several variations of network test problems are generated using NETGEN to investigate the relative performance of the relaxation and primal simplex codes to variations in network density and total supply. Both Bertsekas and Tseng (1988) and Bradley, Brown and Graves (1977) report no significant variations in performance due to cost range variations, to include negative costs; accordingly, cost range variations are not investigated here, and in fact are always kept constant at between 1 and 100 for all test problems. Bradley, Brown and Graves note that their primal simplex codes seem to be more efficient at solving non-random ("real world") networks. However, the difficulty of obtaining suitable non-random networks for test purposes (i.e., widely accepted as appropriate and capable of being reproduced by the mathematical programming community at large) preclude their use in this thesis.

1. Density Variations

Both 400 and 300 node transportation test problems are generated with varying density--up to approximately $(N/2)^2$ and with total supply held constant at 100,000. Tables

A.3 and A.4 contain the results of these tests, and Figure 3.6 summarizes the data in Table A.3.

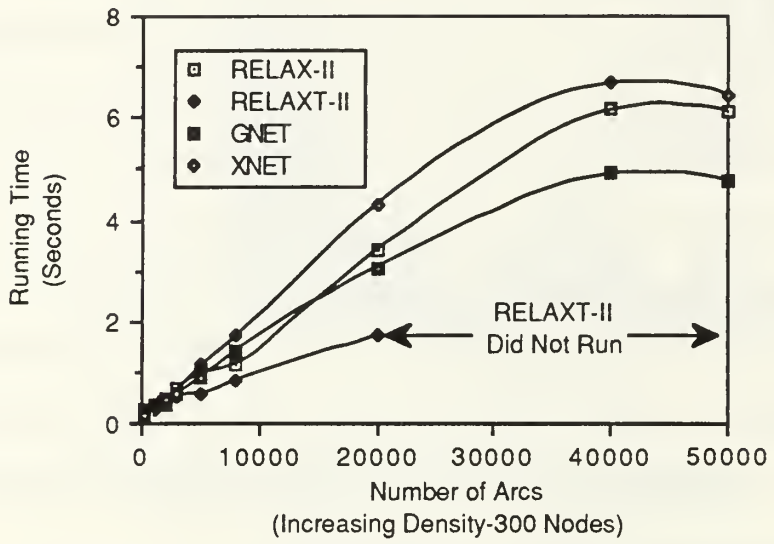

Figure 3.6. Density Variations in a 400 Node Transportation Network

Increasing the density of a network appears to make RELAXT-II even more efficient relative to the other codes, although it again fails to run when solving the higher density problems. In this case the RELAXT-II code never terminates on any solution; execution is halted after about sixty seconds of running time, ten times the running time of the slowest code. Note that GNET becomes more efficient relative to RELAX-II as density increases, by about twenty percent, while XNET and RELAX-II are approximately equal with the relaxation code slightly ahead.

Another density variation is investigated in a 300 node transportation problem, this time with total supply held constant at 150--making the network an assignment problem as produced by NETGEN. Running times can be seen in Table A.5. These results indicate that the relaxation codes maintain their performance edge with changing network density in assignment problems, although RELAXT-II again fails to terminate execution on the highest density test problems.

2. Total Supply Variations

Total supply is varied for two separate transportation test problems: a high density network of 300 nodes and 20,000 arcs, and a low density network of 300 nodes and 2,000 arcs. Running times for both of these experiments can be seen in Tables A.6 and A.7, both of which are summarized in Figures 3.7 and 3.8 respectively.

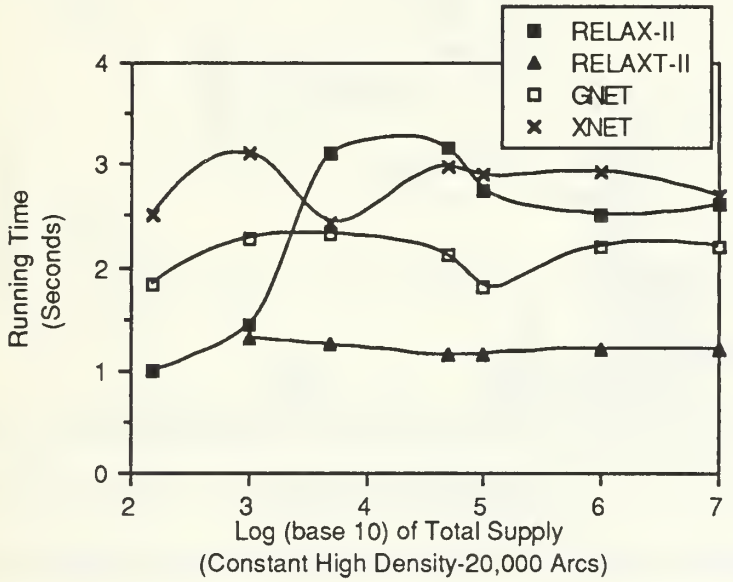

Figure 3.7. Variations in Total Supply (High Density)

Once again the RELAXT-II code fails to terminate with a solution for a high density network; in this case the test problem containing a total supply of 150 units. RELAXT-II does, however, maintain its performance edge across all of the total supply variations. Relative performances seem to be independent of variations of total supply in both high and low density transportation networks, except that total supplies of above 100,000 appear to favor GNET over RELAX-II, while XNET performs on par with RELAX-II.

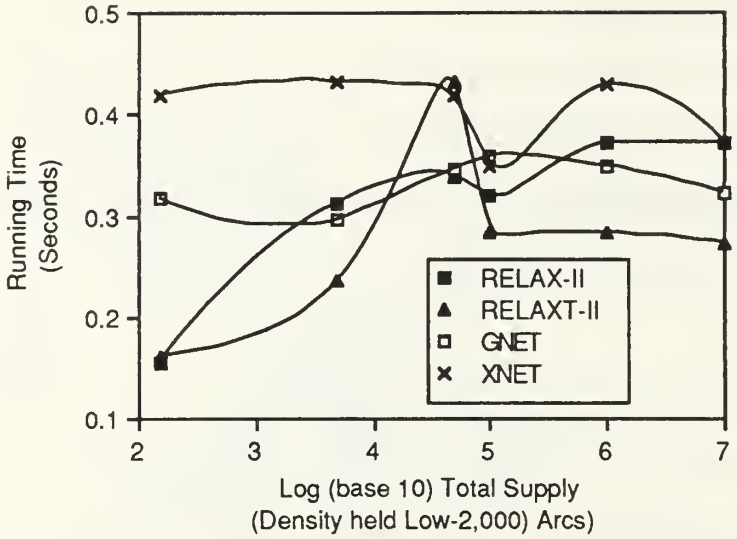

Figure 3.8. Variations in Total Supply (Low Density)

E. KILOBYTE-SECOND ANALYSIS

The storage requirements for the relaxation routines are considerably higher than for the primal simplex routines and are acknowledged by Bertsekas and Tseng to be the main disadvantage of the relaxation methods. While technological trends indicate that computer memory is becoming less expensive, there are still valid reasons for demanding storage efficiency.

When relatively small problems are being solved on large computers, storage is no great consideration. The size (read richness and fidelity) of real-world network problems is often constrained by computer storage limitations, however, not necessarily just speed of computation. Even if a given problem can be feasibly solved with the technology at hand, more detail is often desired which demands not only better solution efficiency, but a smaller storage requirement. Also, if one is limited to solving network problems on a personal computer, as is done today with more frequency, storage requirements can easily be the major limiting factor. In short, there are many realistic cases where storage efficiency may desired ahead of a computational efficiency.

To address these concerns a kilobyte-second analysis is presented. Total storage requirements (compiled program size plus array storage) is determined for each of the four codes being evaluated for each test problem, and is multiplied by the running times for each test problem. Figures 3.9, 3.10 and 3.11 summarize this analysis. VSNET problem number 6 is not included in Figure 3.11 because greatly it distorts the scaling of the graph.

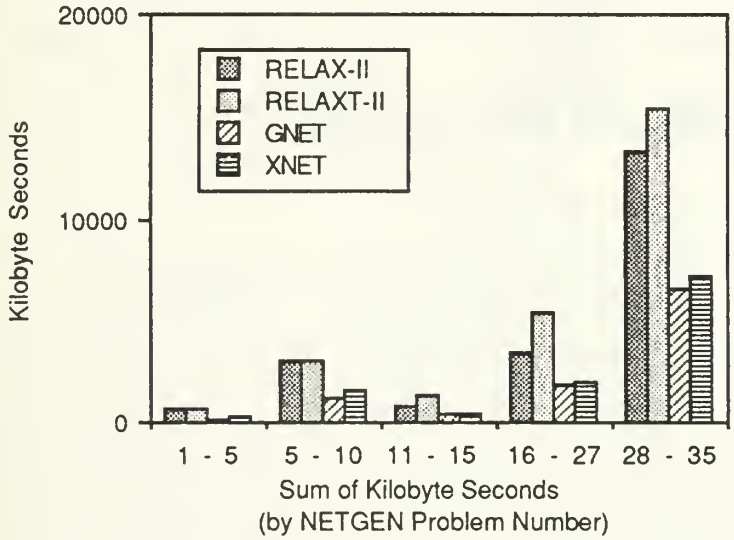

Figure 3.9. Kilobyte-Second Analysis of NETGEN 1-35

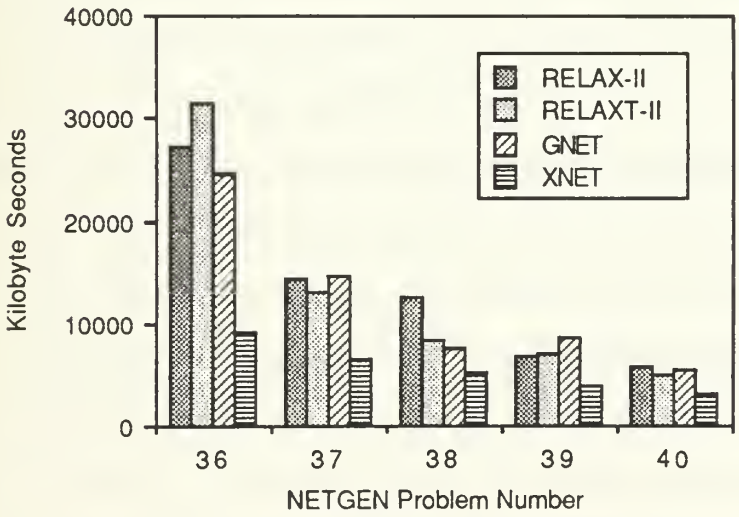

Figure 3.10. Kilobyte-Second Analysis of NETGEN 36-40

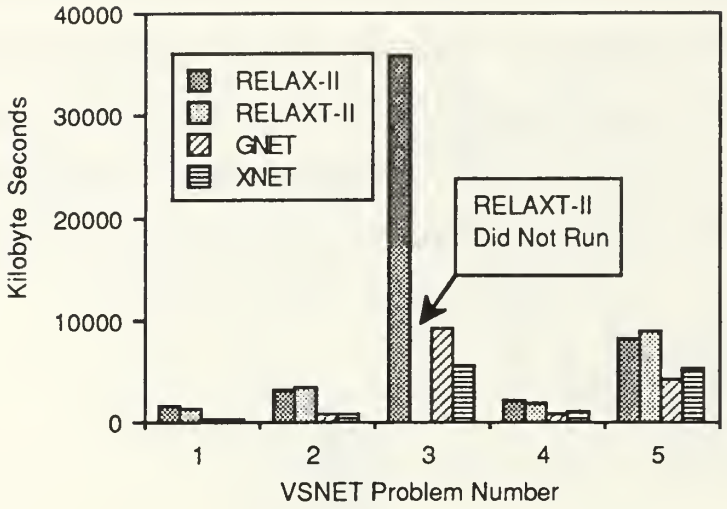

Figure 3.11. Kilobyte-Second Analysis of VSNET 1-5

It is clear that both primal simplex codes perform much better when storage is considered as part of the measure of effectiveness. XNET is particularly good in the large, randomly generated NETGEN problems with many sinks (36-40) and GNET looks quite competitive across the board. There were in fact no network problem categories in which the relaxation codes were competitive within the framework of this measure of effectiveness.

IV. IMPLEMENTATION ASPECTS OF THE RELAXATION METHODOLOGY

This chapter discusses some of the implementation issues of the relaxation methodology. After a description of how Bertsekas and Tseng have designed their codes, several modifications of RELAX-II are put forward and analyzed that reveal promising directions for further research. Emphasis is on the RELAX-II code because it is the most immediately instructive of the two available codes.

A. IMPLEMENTATION OF THE RELAXATION METHOD

The algorithm given at the end of Chapter II can be broken down into a basic flow of actions. First, it is necessary to find some node that has deficit and to place this node into the set of nodes under consideration for a price change (the set S identified in Chapter II). Second, it must be determined whether it is advantageous to change the price of the selected node, or whether it is possible to push flow along a flow augmenting path that begins with the first selected node. Finally, if the dual function cannot be improved via a price change and no flow augmenting path has been found, a decision must be made as to how to add another node to S from all the possible candidates in set L.

Recall that if the process stops before a second node is added to S then a single node iteration (SNI) has been performed; with more than one node included in S, a multiple node iteration (MNI) has been concluded. It is intuitive to expect that a SNI is more efficient than a multiple node iteration, and in fact the computational experience of Bertsekas and Tseng corroborates this observation, to the point where they intentionally try to increase the relative occurrence of SNIs over MNIs in both their implementing codes. In the preprocessing phase of each code--included in the reported running times--each arc capacity

is set to as small a value as possible without changing the optimal solution. For example, in a transportation problem, each arc capacity is set to the minimum of the supply and demand of the head and tail nodes. By tightening the arc capacity the incidence of SNIs tends to increase, although Bertsekas and Tseng do not have a ready explanation for this phenomenon.

A network problem is presented to the algorithms as a simple edge list. Lower bounds are assumed to be zero. If any lower bounds are present in the problem, the user is expected to apply the standard transformation $x'_{ij} = x_{ij} - l_{ij}$, allow the algorithm to solve for x'^*, and then reverse the transformation. The edge list is read by a data input subroutine (to be written by the user) and then transformed into the data structures used by the relaxation codes with a subroutine named INIDAT, provided by Bertsekas and Tseng.

Subroutine INIDAT has inputs of: NA, number of arcs in the network; N, number of nodes; STARTN(j), the array of head nodes of arc j; and ENDN(j), the array of tail nodes of arc j. It produces as output a linked list for each set of incident arcs to each node, both in forward and reverse star forms. The output arrays of INIDAT are: FOU(i), containing the first of the arcs leaving node i; NXTOU(j), the next arc to j leaving STARTN(j); FIN(i), the first arc entering node i; NXTIN(j), the next arc to j entering END(j). FOU and NXTOU constitute the forward star representation of the network, while FIN and NXTIN are the reverse star. Although these arrays are really just a series of pointers, they are an unusual data structure; an example of how they are implemented can be seen in Figure 4.1.

Why has this data structure been selected? To address this question the running times of two simple test programs that perform a depth-first search are observed, each differing only by the type of data structure used: one with a hierarchical adjacency list (HAL), which is used in both primal simplex codes, and the other with a linked list created by INIDAT.

Across a series of test problems, the test program using the linked list is seen to be about fifty percent faster than the program using the HAL data data structure.

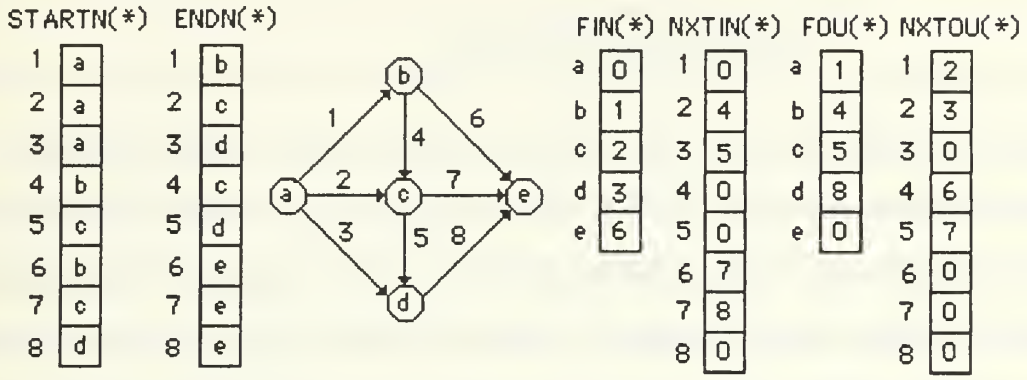

Figure 4.1. An Example of the INIDAT Data Structures.

This dramatic performance difference is attributed to the manner in which the two data structures access adjacent arcs. The HAL data structure uses a $|N|+1$ length array , EP(*), which is an entry pointer into an $|A|$ length array, TAIL(*), which in turn contains the nodes adjacent to some selected node. For example, a code to find all nodes adjacent to node STARTNODE in a reverse star HAL is (variables longer than 6 characters are used for clarity)

```
        DO 100 I=EP(STARTNODE), EP(STARTNODE+1)-1
        ADJACENTNODE=TAIL(I)
        .
        .
        .
100     CONTINUE
```

which assigns the nodes of interest to the variable ADJACENTNODE. Using the FIN(*) and NXTIN(*) arrays described above in a linked list of the type produced by subroutine INIDAT, a code to find the arcs incident to STARTNODE is

```
        ARC=FOU(STARTNODE)
100     .
        .
        .
        ARC=NXTIN(ARC)
        IF(ARC.NE.0) GO TO 100
```

where the variable ARC can be used as an index to access the network data arrays. The observed performance difference of the two data structures is probably due to the fact that the HAL data structure must use a DO loop for which both the starting and ending value of the index variable must be computed. The INIDAT-created linked list data structure is more efficient since it uses only direct assignment statements and one IF check against a constant (zero).

The input parameters for both the RELAX-II and RELAXT-II subroutines contain all of the scalars and arrays that are inputs and outputs of INIDAT, plus array U(j), the flow capacity of arc j, and array B(i), the demand of node i (positive for demand nodes and negative for supply nodes), both of which are read from the input edge list.

At this point the procedures of the two algorithms diverge. The remainder of this section will be devoted to an exploration of the RELAX-II implementation of the relaxation methodology.

RELAX-II initially performs a feasibility check of the network after which the initial prices of all nodes are set to zero. Flows are set to zero for nonnegative arc costs, and to the upper bound for negative arc costs. Once flows are initialized, the starting deficit of each node is calculated and stored in array DFCT.

The stage is now set for the selection of the first node to enter S. This is done by simply selecting the node associated with position one of the node length array DFCT. If the selected node happens to have a deficit, the relaxation method begins an iteration with this first node. Otherwise, the next node in array DFCT is considered in order. If there are

no nodes with deficit remaining (DFCT contains all zeroes), the algorithm terminates. The search procedure is implemented by means of a DO loop, which searches the DFCT array for nodes still having some deficit by repeatedly cycling through the DFCT array, taking the nodes in the order in which they happen to have been entered into the data structure from the original edge list. Once a node with deficit has been found, RELAX-II will attempt to perform a SNI with this node. If it is not possible to do either a dual function ascent or a flow augmentation with the first selected node, more nodes adjacent to the starting node will be allowed to enter S, as outlined in the algorithm in Chapter II. This process of increasing the number of nodes in S will continue incrementally until certain stopping criteria discussed below are met.

The opening strategy of RELAX-II is to temporarily limit all iterations to be SNIs. This is done by not allowing any MNIs to occur during the first two loops through the DFCT array, i.e., if the selected single node fails to produce a dual function improvement or a flow augmentation, the iteration attempt will terminate before any more nodes are added to S, the DO loop counter will increment by one, and the next position in the DFCT array will be checked for a node possessing a deficit. The purpose of this opening procedure is to attempt to phase in as much initial flow as possible with cheap SNIs. This strategy works well for all problems, but it is especially beneficial for transportation and assignment problems.

Once two full loops though the DFCT array have been made, MNIs are allowed in conjunction with SNIs. Specifically, if a SNI attempt has proven unsuccessful, then more nodes are allowed to enter set S. If the SNI is successful then the iteration terminates, the DO loop counter is incremented and the next node in the DFCT array is considered. In this way, MNIs only occur as a consequence of a failed SNI. Nodes will continue to be placed in S until the residual capacity across the cut of S is less than the total deficit of all nodes in

S. (The residual capacity of S is the difference between the current flow of all arcs crossing the set boundary and the upper bounds of these arcs.)

In certain specific cases, an adaptive strategy is imposed to control the occurrence of MNI price changes by the use of two scalars: TP and TS. When the total number of nodes with deficit is less than TP, and S has grown to include a total of TS nodes, no MNI price change is allowed, although flow augmentation may still occur. After much experimentation, Bertsekas and Tseng have set TP to a value of 10 and TS is to $|N|/15$, and report that these values seem to provide the best all around performance for RELAX-II.

B. EXPERIMENTS INVOLVING SORTED INPUT

As stated, the relaxation codes read the network data in the order in which it is presented in the edge list; this initial ordering of the input data is important to the priority in which nodes are considered for initializing an iteration step because it forces the ordering of DFCT. Is there a better way to present the data to the algorithm? To investigate this question four different sortings of the input data are investigated: by ascending and descending arc cost values, and by ascending and descending arc capacity values. Sorting is accomplished before the actual relaxation codes are given the problem, and are not included in the running time results shown in Tables B.1 and B.2 for the RELAX-II and RELAXT-II codes respectively. Figures 4.2 and 4.3 summarize Tables B.1 and B.2 with the percentage of change in running times expressed in terms of the total running time for all forty standard NETGEN problems.

There is a small advantage to pre-sorting data for these codes, particularly by descending arc capacity, but not a large one. Clearly, it may not be worthwhile to expend computer resources sorting data for a small network such as those that are used for testing here--the time to sort the data would be longer than the savings gained. However, there are two situations in which a sort may prove useful. First, if a problem is large enough, the

employment of an efficient sorting scheme may be advantageous. Second, if a network is to be solved many times, without a change in the network structure, a single initial sort may produce a substantial time savings over multiple runs.

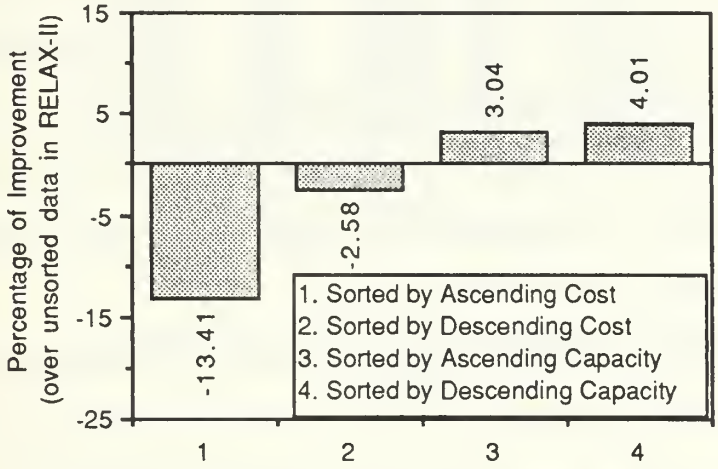

Figure 4.2. Data Pre-sort Effect on Running Time (RELAX-II)

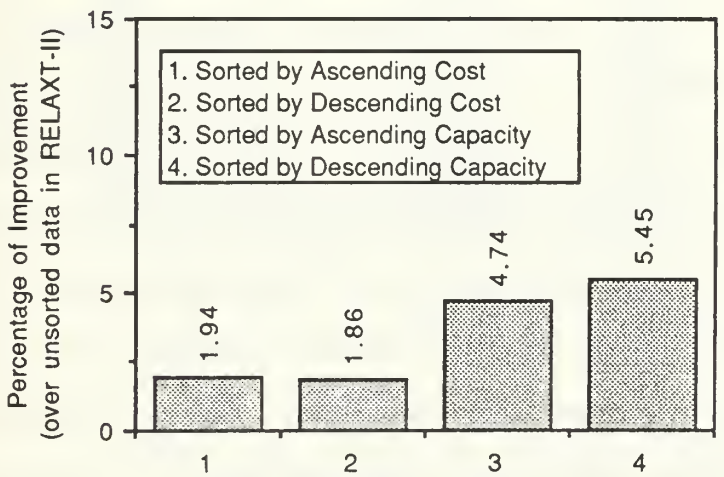

Figure 4.3. Data Pre-sort Effect on Running Time (RELAXT-II)

C. DYNAMIC PRIORITY QUEUE MODIFICATION

When determining which node to select as the starting node for an iteration, RELAX-II does not employ any ranking scheme, it merely considers nodes in the order in which they are presented by the original edge list. A modification of RELAX-II is investigated in which the node selection process is based on a node's absolute deficit. By employing a dynamic priority queue of node deficits, it is possible to always select the node with the maximum absolute deficit for consideration as a starting node in any iteration. This is done by continually updating the priority queue every time a node deficit is changed, and then selecting the leading member of the queue when the next iteration is to begin. The specific priority queue used is a *binary tree*, also called a *two-heap*.

To implement the priority queue in RELAX-II, the DO loop that controls the search of the DFCT array is eliminated from the code and a queue selection process is substituted. As modifications to node deficits occurred during the relaxation iteration process, the binary tree is continually updated. Thus, it is possible to identify the precise order of absolute deficits at any point in the algorithm, a completely dynamic priority queue.

Baseline iteration counts were conducted to evaluate the number of single and multiple node iterations (SNIs and MNIs respectively) performed by the unmodified RELAX-II code, for both the NETGEN and VSNET problem sets . These can be seen in Table B.3. Table B.4 contains the number of SNIs and MNIs performed by RELAX-II (PQ), and the associated running times. The running time behavior for RELAX-II(PQ) is not impressive (Appendix B.4), particularly for assignment problems, but the number of SNIs and MNIs performed has been dramatically reduced. Figure 4.4 shows the savings in SNIs made for the first 35 NETGEN problems; Figure 4.5 the savings in SNIs made for NETGEN 36-40 and the VSNET problem set; Figure 4.6 the savings in MNIs made for the first 35 NETGEN problems; and Figure 4.7 the savings made for NETGEN 36-40 and the VSNET problem set.

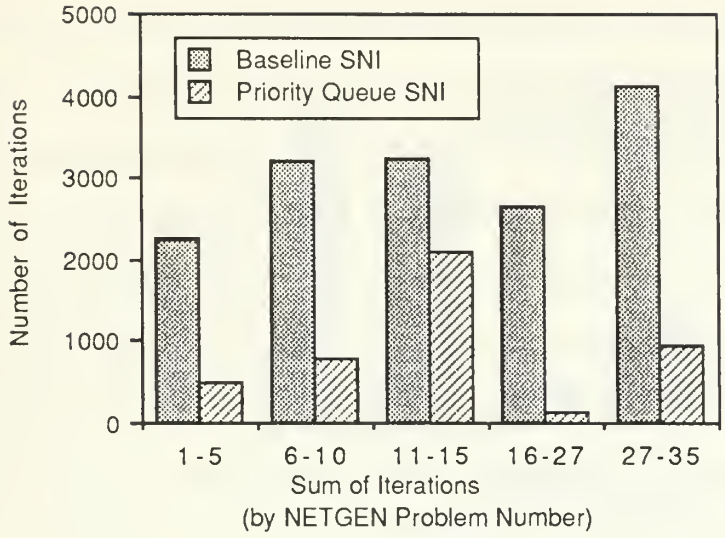

Figure 4.4. Single Node Iteration Savings NETGEN 1-35

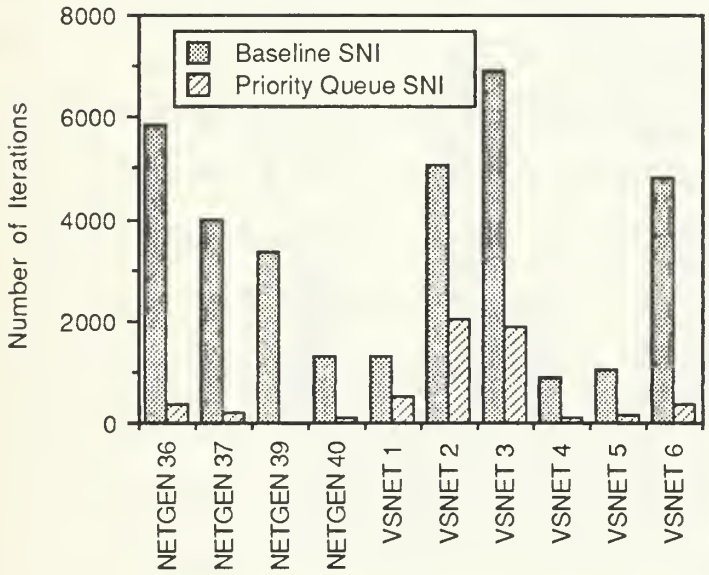

Figure 4.5. Single Node Iteration Savings NETGEN 36-40 and VSNET 1-6

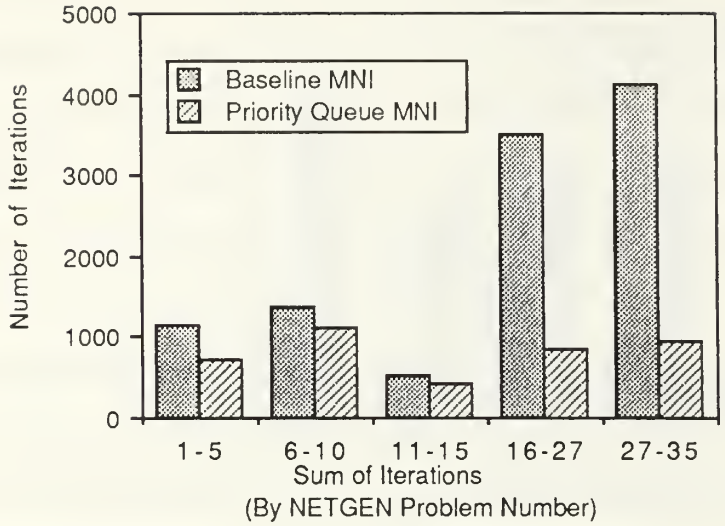

Figure 4.6. Multiple Node Iteration Savings NETGEN 1-35

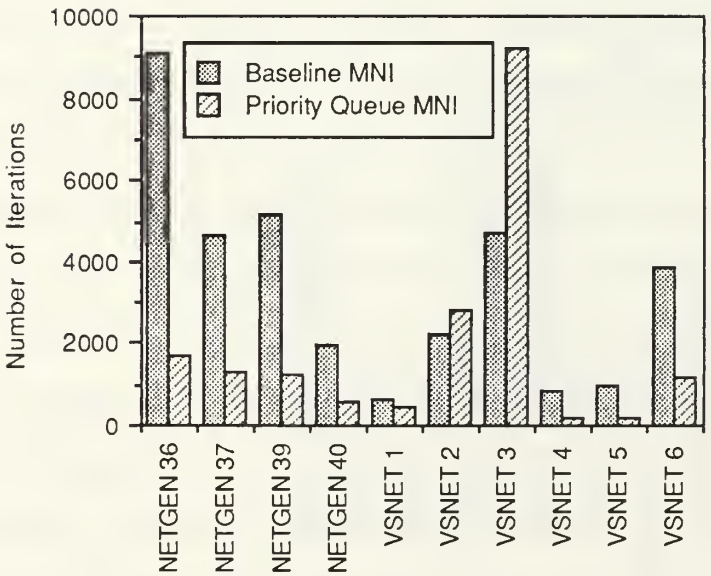

Figure 4.7. Multiple Node Iteration Savings NETGEN 36-40 and VSNET 1-6

Prioritizing nodes by absolute deficit clearly reduces the number of iterations, both SNIs and MNIs. It is also clear that prioritization does not improve the running time

performance of RELAX-II, at least as implemented by a priority queue. The natural question is: does prioritizing the node selection process somehow increase the time per iteration over the unmodified version of RELAX-II? To investigate this question a timing function is installed in both RELAX-II and RELAX-II(PQ) to measure the duration of the average SNI and MNI. Implementing the measurement requires calls to be made to a timing function in dozens of locations throughout the program which necessarily confounds measurement accuracy so that the results must be treated with some skepticism. The tests reveal, however, that while RELAX-II(PQ) produces less efficient MNIs, it reduces the running time of the average SNI. Referring to Figures 4.4 through 4.7, the usual RELAX-II(PQ) savings in MNIs is about 50 percent, while the savings in SNIs is much more than this, a factor of five or more in all test networks except assignment problems. Note also from Table B.3 and B.4 that SNIs are always much more numerous that MNIs.

The indication is that prioritizing the node selection process may yield a net savings in running time; exactly how much it is impossible to say because the performance of the timing experiment was not satisfactory. In any case, whatever computational savings are being generated by the node prioritization process are being compromised by the inefficiencies of the priority queue implementation.

A variation of the dynamic priority queue is explored in which array DFCT is ordered a single time by absolute node deficit before the relaxation process is allowed to begin. After this first sort, which is not included in the running time measurements, the relaxation process is allowed to proceed as in the original, unmodified version to find the optimal solution. Running times (see Table B.5) for this variation are not improved for the NETGEN transportation, assignment and capacitated transshipment problems. There is, however, a measurable improvement in the large scale NETGEN (about 3 percent) and

VSNET (about 4 percent) problems, indicating that even a single initial sorting of nodes in order of decreasing absolute deficit produces a visible change in performance.

D. PARTIAL SORT VARIATION

A partial sort variation of RELAX-II is implemented as follows. First, the DFCT array is compressed by removing all zero deficit node elements, and a pointer array POINT is created that identifies which node is associated which each deficit in DFCT. (If POINT(n) is zero then node n is not on array DFCT, otherwise, POINT(n) identifies the node associated with a position in DFCT.) Next, a variable LAST that identifies the current last position of DFCT, and a variable CPOS that holds the current position in DFCT under consideration are created. As nodes develop nonzero deficits during the relaxation process, they are placed at the end of the DFCT array and LAST and POINT are updated. As node deficits become zero, say DFCT(n)=0, the assignments DFCT(n)=DFCT(LAST) are made, and LAST is reduced by one and POINT is updated. The DO loop is allowed to cycle CPOS from one to LAST repeatedly, selecting nodes for relaxation iterations until LAST=0, i.e., all nodes have zero deficit. This procedure will create a relatively efficient, but unordered, cycling through the current nodes with deficits.

Having the above data structure, when the node associated with DFCT(CPOS) is being considered as a starting node in a relaxation iteration, all nodes from DFCT(CPOS) to DFCT(CPOS+NTT) are searched to find the node with the maximum absolute deficit. If CPOS+NTT>LAST, the search is conducted from DFCT(LAST-NTT) to DFCT(LAST). If LAST<NTT, the search is conducted from DFCT(1) to DFCT(LAST). In this way a "local" maximum absolute deficit will be found, but with much less computational effort than that required to accomplish a dynamic priority queue sort. The problem then becomes one of finding the optimum setting of NTT for the network problem to be solved.

Experimental runs were conducted for all forty standard NETGEN problems and the six specially constructed VSNET problems. For each problem, NTT is varied until a minimum running time is obtained; results are contained in Table B.6 and Figures 4.8 and 4.9 summarize these data. Two particulars stand out. First, missing the optimum value of NTT by as little as one unit caused, for some problems, a massive increase in running time. Second, the optimum NTT value could not be related to any network parameter such as number of nodes, number of arcs, cost range, etc. Thus, the partial sort modification of RELAX-II, RELAX-II(PS), is not useful in practice since each problem must be run repetitively until an optimum NTT value can be found. It does show, however, that there are some significant improvements to be gained even by partially prioritizing the node selection process in the relaxation algorithms.

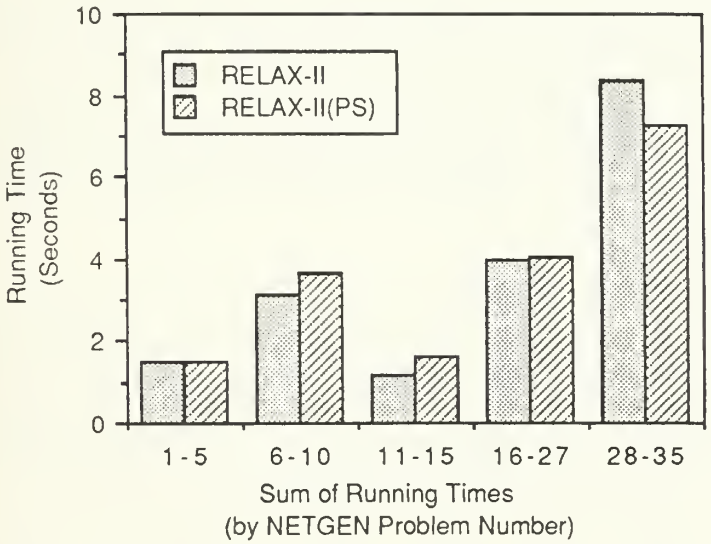

Figure 4.8. RELAX-II and RELAX-II(PS) Running Time Comparison (NETGEN 1-35)

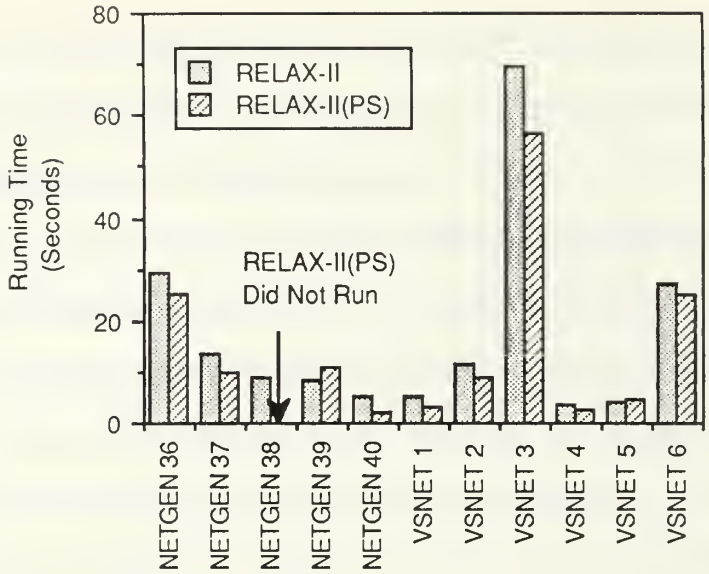

Figure 4.9. RELAX-II and RELAX-II(PS) Running Time Comparison (NETGEN 36-40 and VSNET 1-6)

RELAX-II(PS) is equal to or worse than RELAX-II when solving transportation, assignment, and capacitated transshipment problems, but produces substantial run time efficiencies for uncapacitated transshipment problems and large scale problems in the NETGEN problems set. In the VSNET problem set, RELAX-II(PS) ran more efficiently in every case except problem number five.

Clearly, there is an advantage to prioritizing starting nodes when using the relaxation method. RELAX-II(PQ) shows that there is a substantial iteration savings associated with selecting relaxation starting nodes by maximum absolute deficit, although the binary tree sort is perhaps not the way to implement the idea. RELAX-II(PS) shows that prioritizing over some small subset of nodes with deficit produces good results, but obvious implementation problems are apparent. Is there some method of applying a prioritization scheme to the node selection process in the relaxation methodology that is both computationally efficient and can be effectively implemented? Further research could potentially produce a better version of RELAX-II.

V. CONCLUSIONS

This thesis primarily investigates the relative computational efficiencies of the original, unmodified primal simplex codes GNET and XNET, and the newer relaxation codes RELAX-II and RELAXT-II. Tests conducted by Bertsekas and Tseng (1988) had shown the relaxation codes to be on the order of four to five times faster than the primal simplex codes at solving the standard NETGEN problem set. Their results are not duplicated here. Within the limits of the testing method employed, it is found that while the relaxation codes do perform better when solving specific standard NETGEN problems (1-35), the primal simplex codes are still closely competitive. In particular, the primal simplex codes are more efficient at solving large, randomly generated problems with many more sinks than sources (NETGEN 36-40) and capacitated transshipment networks that contain something other than a purely random structure. The following specific assessments are made.

1. Table 5.1 summarizes the running times of the four codes evaluated for all test problems. When VSNET 3, which RELAXT-II did not solve, is included in the total sum of running times for all test problems, both GNET and XNET perform faster than RELAX-II. When VSNET 3 is excluded from the total sum of running times, RELAXT-II is the fastest code, but XNET remains faster than RELAX-II in this and all other cases. These results do not duplicate those obtained by Bertsekas and Tseng (1988) in their evaluation of the relaxation codes against a primal simplex code developed by Grigoriadis and Hsu (1980), and in fact show that primal simplex methods remain closely competitive.

2. Both relaxation codes are more computationally efficient than the primal simplex codes at solving transportation and assignment problems in all observed cases.

3. The primal simplex codes tend to be faster when solving VSNET-generated transshipment networks, particularly when capacitated. This is attributed to the heuristics

incorporated in both GNET and XNET that take advantage of networks that have an echelon structure with more sinks than sources.

TABLE 5.1. SUMMARY OF RUNNING TIMES FOR ALL TEST PROBLEMS

Row	Problem Set	RELAX-II	RELAXT-II	GNET	XNET
1	Total NETGEN	85.33	71.09	153.52	79.74
2	Total VSNET (#3 Not Incl.)	120.63	49.18	97.94	114.73
3	Total VSNET (#3 Incl.)	147.89	NA	114.41	124.38
4	Sum of Row1 and Row2	233.20	NA	267.93	204.12
5	Sum of Row1 and Row3	205.96	120.27	251.46	194.47

4. RELAXT-II clearly runs fastest on uncapacitated transshipment problems in all cases when it runs. RELAX-II is less computationally efficient in these problems, and is comparable to the primal simplex codes, particularly on those problems generated by VSNET. Again, the network structure characteristics that are created by VSNET seem to favor the primal simplex codes.

5. Increases in transportation problem density tends to favor the primal simplex codes, at least when compared to RELAX-II. RELAXT-II did not run for the higher density networks, although it is clearly the fastest code for the low and moderate density networks.

6. The relaxation codes maintain a clear performance edge in all density variations of assignment problems, although RELAXT-II again failed to run at high density.

7. When varying total supply in transportation problems, both relaxation codes are superior when total supply is set below 1,000, while GNET is usually superior to RELAX-

II when total supply is set above 5,000 (see Tables A.6 and A.7). This is particularly true for high density networks, but is also observed to a lesser extent in low density networks.

8. The primal simplex codes typically require less than half the computer storage space that the relaxation codes require. This represents a serious limitation to the analyst interested in solving large scale networks on a mainframe computer, or smaller networks on a personal computer. In all cases, the primal simplex codes are more efficient when running time and storage requirements are combined into a single measure of effectiveness.

9. It can be advantageous to sort the edge list data input to both relaxation codes prior to commencing the solution process. Sorting by descending arc capacity gives the best results. Sorting the edge list in this manner is expected to be fruitful for very large networks when a very efficient sorting routine is used, or when a network is to be solved repetitively without a change in the network parameters.

10. Implementing a dynamic priority queue to select the node with the largest absolute deficit as a starting node in a relaxation iteration dramatically cuts down on the number of single and multiple node iterations. While the resulting multiple node iterations appear to be slower, single node iterations appear to be faster. With single node iterations greatly outnumbering multiple node iterations, a net improvement in running time efficiency could be expected. However, the priority queue used in this thesis to implement RELAX-II(PQ) is not efficient enough to take advantage of the inherent time savings. More research is needed.

11. A single sort variation of the priority queue was investigated and found to have no effect on transportation, assignment and capacitated transshipment problems. Modest improvements in computer running time are observed for the large scale NETGEN problems and for most of the VSNET problems.

12. A partial sort variation RELAX-II(PS) showed no improvement when solving the NETGEN transportation, assignment or capacitated transshipment problems, but did produce substantial efficiencies for NETGEN uncapacitated transshipment and the VSNET networks. The implementation difficulties inherent in RELAX-II(PS) tend to negate its advantages since each problem must be specially manipulated to attain any run-time gains. This modification does, however, illustrate that there are advantages to be gained even by modest prioritization of deficits.

The research has produced compelling evidence that the relaxation algorithms can be further refined, possibly to the point where they are clearly superior to the current primal simplex codes. All indications reinforce the idea of prioritizing the node selection process by absolute deficit. Unfortunately, the modifications attempted in this thesis proved to be less efficient in total running time than the original codes. It is strongly suspected that a solution algorithm that incorporates the basic relaxation method, in conjunction with some kind of intelligent node selection process, will prove highly efficient. The actual design of such an algorithm is beyond the scope of this thesis and is consequently left as a direction for further research.

APPENDIX A UNMODIFIED RUNNING TIMES

This Appendix contains the unmodified running times of several experimental runs of the network solvers RELAX-II, RELAXT-II, GNET and XNET. All experiments were conducted on an IBM 3033AP computer employing CMS version 5.0 and operating on a time share basis. The solver codes were compiled by VS Fortran version 1.4.1 under optimization level 3.

TABLE A.1. NETGEN STANDARD PROBLEM SET

These standard benchmark test problems are from Klingman, Napier and Stutz (1974) and were obtained using the network generator NETGEN. Running times are in seconds.

NETGEN#	RELAXII	RELAXTII	GNET	XNET	
1	0.18	0.19	0.19	0.24	
2	0.26	0.21	0.22	0.24	
3	0.26	0.20	0.27	0.32	200 Node Transport
4	0.48	0.33	0.25	0.34	
5	0.33	0.26	0.33	0.36	
Sum	1.51	1.20	1.26	1.49	
6	0.48	0.46	0.43	0.56	
7	0.61	0.47	0.62	0.80	
8	0.71	0.52	0.82	0.99	300 Node Transport
9	0.63	0.43	0.84	1.09	
10	0.71	0.49	0.94	1.17	
Sum	3.14	2.36	3.65	4.60	
11	0.12	0.12	0.39	0.45	
12	0.14	0.17	0.46	0.60	
13	0.24	0.29	0.48	0.68	400 Node Assignment
14	0.23	0.30	0.66	0.70	
15	0.45	0.39	0.88	0.83	
Sum	1.19	1.27	2.88	3.24	

TABLE A.1. (CONTINUED)

NETGEN#	RELAXII	RELAXTII	GNET	XNET	
16	0.26	0.27	0.28	0.35	
17	0.37	0.43	0.38	0.41	
18	0.31	0.27	0.31	0.32	
19	0.34	0.38	0.42	0.43	
20	0.34	0.36	0.30	0.33	
21	0.43	0.33	0.31	0.34	400 Node Capacitated
22	0.32	0.37	0.27	0.28	Networks
23	0.60	0.53	0.31	0.29	
24	0.18	0.17	0.29	0.30	
25	0.46	0.39	0.34	0.34	
26	0.13	0.11	0.26	0.22	
27	0.26	0.26	0.27	0.27	
Sum	4.01	3.86	3.76	3.88	
28	0.73	0.59	1.27	0.82	
29	0.81	0.75	1.31	0.83	
30	1.25	1.02	1.37	0.92	
31	0.84	0.72	1.38	1.00	1000 and 1500 Node
32	1.11	1.11	2.40	1.27	Uncapacitated Networks
33	1.49	1.26	2.41	1.27	
34	0.82	0.84	2.57	1.46	
35	1.31	1.47	2.56	1.49	
Sum	8.35	7.76	15.27	9.05	
36	30.24	29.19	51.47	17.89	
37	13.68	9.83	25.71	13.54	
38	8.99	4.57	14.30	9.68	Large Networks
39	8.72	7.38	23.49	9.80	
40	5.51	3.67	11.73	6.57	
Sum	67.13	54.64	126.70	57.48	

TABLE A.2. VSNET PROBLEM SET

Additional test problems obtained by using VSNET, a network problem generator developed by Bonwit (1984) at the Naval Postgraduate School, Monterey, California. VSNET creates transshipment networks as a series of echelons (specified by the user) upon which a set of random arcs are constructed. Problem generation parameters were all held constant except for those indicated in the table. All running times are in seconds.

VSNET#	#Nodes	#Arcs	RELAXII	RELAXTII	GNET	XNET
Capacitated Transshipment Networks						
1	500	10000	3.75	2.27	1.34	1.38
2	1000	20000	4.04	3.24	2.71	2.71
3	5000	30000	27.26	DNR	16.47	9.65
Sum			35.05	5.51	20.52	13.74
Uncapacitated Transshipment Netwoks						
4	500	10000	4.98	3.24	5.96	7.49
5	1000	18000	11.42	9.55	16.28	19.23
6	5000	30000	96.44	30.88	71.66	83.92
Sum			112.84	43.67	93.89	110.64

TABLE A.3. INCREASING NETWORK DENSITY (400 NODE)

Variations on the NETGEN problem set. These networks are 400 node transportation problems, all with a constant total supply of 100,000 and a cost range parameter of 1 to 100. Density has been varied up to approximately $(N/2)^2$. All running times are in seconds.

#Nodes	#Arcs	RELAXII	RELAXTII	GNET	XNET
400	200	0.17	0.13	0.24	0.17
400	1200	0.36	0.27	0.39	0.36
400	1800	0.38	0.37	0.43	0.43
400	2000	0.38	0.38	0.47	0.48
400	3000	0.67	0.55	0.57	0.59
400	5000	0.98	0.57	0.88	1.13
400	8000	1.13	0.82	1.43	1.70
400	20000	3.39	1.70	3.05	4.31
400	40000	6.16	DNR	4.90	6.68
400	50000	6.12	DNR	4.77	6.45

DNR: Did Not Run

TABLE A.4. INCREASING NETWORK DENSITY (300 NODE)

More variations on the NETGEN problem set. These networks are 300 node transportation problems, all with a constant total supply of 100,000 and a cost range parameter of 1 to 100. Density has been varied up to approximately $(N/2)^2$. All running times are in seconds.

#Nodes	#Arcs	RELAXII	RELAXTII	GNET	XNET
300	200	0.10	0.12	0.15	0.11
300	2000	0.32	0.30	0.34	0.41
300	4000	0.54	0.35	0.59	0.61
300	6000	0.83	0.58	0.89	1.10
300	10000	1.49	0.85	1.23	1.43
300	12000	2.11	0.95	1.62	2.14
300	16000	1.73	0.99	1.98	2.61
300	18000	2.54	1.18	2.08	2.74
300	20000	2.76	1.27	1.84	2.73
300	22000	2.08	DNR	2.18	2.84

DNR: Did Not Run

TABLE A.5. INCREASING DENSITY NETWORK (300 NODE)

More variations on the NETGEN problem set. These networks are 300 node transportation problems, all with a constant total supply of 150 and a cost range parameter of 1 to 100. Density has been varied up to approximately $(N/2)^2$. All running times are in seconds.

#Nodes	#Arcs	RELAXII	RELAXTII	GNET	XNET
300	200	0.02	0.02	0.06	0.03
300	2000	0.15	0.16	0.36	0.42
300	4000	0.16	0.31	0.34	0.65
300	6000	0.54	0.52	0.93	0.99
300	10000	0.70	0.49	1.41	1.09
300	12000	0.62	0.66	1.48	20.57
300	16000	0.83	0.80	1.86	2.31
300	18000	1.00	DNR	2.25	2.24
300	20000	1.00	DNR	1.83	2.51
300	22000	1.52	DNR	1.93	2.78

DNR: Did Not Run

TABLE A.6. INCREASING TOTAL SUPPLY (CONSTANT HIGH DENSITY)

In this problem set a 300 node transportation problem of a constant high density (20,000 arcs) was created with NETGEN and total supply was allowed to vary. The cost range parameter remained at 1 to 100 and all running times are in seconds.

#Nodes	#Arcs	Supply	RELAXII	RELAXTII	GNET	XNET
300	20000	150	1.00	DNR	1.83	2.51
300	20000	1000	1.43	1.31	2.28	3.12
300	20000	5000	3.12	1.25	2.33	2.44
300	20000	50000	3.16	1.14	2.11	2.98
300	20000	100000	2.75	1.16	1.80	2.90
300	20000	1000000	2.52	1.21	2.21	2.93
300	20000	10000000	2.61	1.22	2.20	2.70

DNR: Did Not Run

TABLE A.7. INCREASING TOTAL SUPPLY (CONSTANT LOW DENSITY)

This problem set contains a 300 node transportation problem of a constant low density (2,000 arcs), also created with NETGEN. Total supply was allowed to vary and the cost range parameter was held constant at between 1 to 100. All running times are in seconds.

#Nodes	#Arcs	Supply	RELAXII	RELAXTII	GNET	XNET
300	20000	150	0.16	0.16	0.32	0.42
300	20000	5000	0.31	0.24	0.30	0.43
300	20000	50000	0.34	0.43	0.35	0.42
300	20000	100000	0.32	0.28	0.36	0.35
300	20000	1000000	0.37	0.28	0.35	0.43
300	20000	10000000	0.37	0.27	0.32	0.37

APPENDIX B MODIFIED RUNNING TIMES

This Appendix contains the solution times and iteration count data for several experimental versions of the relaxation algorithms solving NETGEN and VSNET network problems. All experiments were conducted on an IBM 3033AP computer employing CMS version 5.0 and operating on a time share basis. The solver codes were compiled by VS Fortran version 1.4.1 under optimization level 3.

TABLE B.1. SORTING INPUT FOR RELAX-II

This table contains running times obtained by pre-sorting network data before allowing the unmodified RELAX-II code to solve the problem. The test problems are the standard set of forty test problems generated by NETGEN.

NETGEN#	RELAXII (baseline)	Ascending Cost	Descending Cost	Ascending Capacity	Descending Capacity
1	0.18	0.15	0.18	0.18	0.18
2	0.26	0.26	0.21	0.26	0.27
3	0.26	0.25	0.29	0.28	0.25
4	0.48	0.46	0.31	0.43	0.44
5	0.33	0.30	0.33	0.31	0.30
Sum	1.51	1.42	1.33	1.46	1.43
6	0.48	0.61	0.48	0.46	0.47
7	0.61	0.71	0.54	0.55	0.57
8	0.71	0.60	0.67	0.65	0.64
9	0.63	0.77	0.73	0.57	0.59
10	0.71	0.75	0.78	0.69	0.69
Sum	3.14	3.44	3.21	2.90	2.96
11	0.12	0.13	0.10	0.11	0.14
12	0.14	0.15	0.18	0.14	0.14
13	0.24	0.23	0.22	0.22	0.20
14	0.23	0.26	0.24	0.23	0.20
15	0.45	0.41	0.36	0.43	0.40
Sum	1.19	1.17	1.11	1.13	1.08

TABLE B.1. (CONTINUED)

NETGEN#	RELAXII (baseline)	Ascending Cost	Descending Cost	Ascending Capacity	Descending Capacity
16	0.26	0.26	0.24	0.23	0.22
17	0.37	0.39	0.46	0.39	0.39
18	0.31	0.30	0.27	0.30	0.31
19	0.34	0.34	0.40	0.35	0.38
20	0.34	0.32	0.32	0.35	0.37
21	0.43	0.44	0.43	0.52	0.53
22	0.32	0.32	0.35	0.32	0.32
23	0.60	0.48	0.54	0.46	0.45
24	0.18	0.20	0.21	0.15	0.18
25	0.46	0.48	0.42	0.45	0.48
26	0.13	0.14	0.08	0.11	0.11
27	0.26	0.29	0.28	0.29	0.27
Sum	4.01	3.95	3.98	3.92	3.98
28	0.73	0.54	0.66	0.68	0.62
29	0.81	0.70	0.76	0.74	0.76
30	1.25	1.16	1.08	1.20	1.25
31	0.84	0.77	0.80	0.81	0.85
32	1.11	1.20	1.25	1.06	1.07
33	1.49	1.36	1.29	1.36	1.37
34	0.82	0.82	0.89	0.80	0.80
35	1.31	1.13	1.19	1.24	1.24
Sum	8.35	7.65	7.91	7.89	7.96
36	30.24	34.49	30.62	29.47	29.19
37	13.68	13.43	13.50	13.48	13.21
38	*8.99	DNR	DNR	DNR	DNR
39	8.72	15.98	10.08	8.65	8.46
40	5.51	6.62	6.60	5.17	5.12
Sum	58.14	70.51	60.81	56.77	55.98

DNR: Did Not Run

* Running time not included in sum calculation

TABLE B.2. SORTING INPUT FOR RELAXT-II

This table contains running times obtained by pre-sorting network data before allowing the unmodified RELAXT-II code to solve the problem. The test problems are the standard set of forty test problems generated by NETGEN.

NETGEN#	RELAXTII (baseline)	Ascending Cost	Descending Cost	Ascending Capacity	Descending Capacity
1	0.19	0.17	0.14	0.20	0.18
2	0.21	0.22	0.23	0.17	0.19
3	0.20	0.23	0.20	0.24	0.23
4	0.33	0.30	0.34	0.28	0.30
5	0.26	0.27	0.32	0.25	0.23
Sum	1.20	1.19	1.23	1.15	1.13
6	0.46	0.43	0.46	0.42	0.43
7	0.47	0.44	0.43	0.43	0.43
8	0.52	0.45	0.48	0.50	0.48
9	0.43	0.42	0.45	0.42	0.43
10	0.49	0.25	0.51	0.47	0.49
Sum	2.36	1.98	2.32	2.24	2.26
11	0.12	0.13	0.10	0.10	0.13
12	0.17	0.15	0.14	0.17	0.15
13	0.29	0.22	0.22	0.24	0.23
14	0.30	0.31	0.28	0.29	0.27
15	0.39	0.36	0.39	0.36	0.39
Sum	1.27	1.18	1.13	1.15	1.15

TABLE B.2. (CONTINUED)

NETGEN#	RELAXTII (baseline)	Ascending Cost	Descending Cost	Ascending Capacity	Descending Capacity
16	0.27	0.28	0.26	0.23	0.28
17	0.43	0.48	0.38	0.40	0.41
18	0.27	0.35	0.25	0.22	0.25
19	0.38	0.35	0.34	0.34	0.34
20	0.36	0.38	0.37	0.32	0.31
21	0.33	0.32	0.31	0.34	0.32
22	0.37	0.35	0.36	0.35	0.37
23	0.53	0.51	0.50	0.51	0.51
24	0.17	0.16	0.22	0.16	0.18
25	0.39	0.46	0.37	0.39	0.41
26	0.11	0.10	0.11	0.09	0.11
27	0.26	0.26	0.23	0.24	0.26
Sum	3.86	3.98	3.69	3.58	3.74
28	0.59	0.58	0.69	0.58	0.56
29	0.75	0.75	0.75	0.74	0.71
30	1.02	1.00	1.18	0.95	0.97
31	0.72	0.71	0.71	0.70	0.70
32	1.11	1.21	1.18	1.11	1.10
33	1.26	1.32	1.27	1.23	1.25
34	0.84	0.82	0.80	0.80	0.80
35	1.47	1.13	1.54	1.40	1.37
Sum	7.76	7.52	8.12	7.51	7.46
36	29.19	27.08	26.34	27.92	27.48
37	*9.83	DNR	DNR	DNR	DNR
38	*4.57	DNR	DNR	DNR	DNR
39	7.38	9.06	9.22	7.06	7.05
40	*3.671	*3.684	DNR	*3.474	*3.519
Sum	36.57	36.15	35.56	34.98	34.53

DNR: Did Not Run

* Running time not included in sum calculations

TABLE B.3. BASELINE ITERATION COUNT

This table contains the baseline counts of both single node iterations (SNI) and multiple node iterations (MNI). These data were obtained from the unmodified RELAX-II implementation with only appropriate counting variables added to the code.

NETGEN Num.	Single Node Iterations	Mult. Node Iterations
1	402	218
2	464	292
3	439	215
4	463	271
5	452	157
Sum	2220	1153
6	639	329
7	665	286
8	676	306
9	598	217
10	613	233
Sum	3191	1371
11	604	95
12	592	92
13	674	114
14	661	106
15	704	123
Sum	3235	530
16	351	349
17	306	262
18	323	299
19	302	268
20	205	354
21	208	324
22	202	384
23	195	349
24	134	242
25	178	317
26	111	166
27	122	197
Sum	2637	3511

NETGEN Num.	Single Node Iterations	Mult. Node Iterations
28	427	588
29	410	565
30	356	500
31	394	589
32	638	901
33	677	1054
34	574	762
35	631	912
Sum	4107	4107
36	5835	9081
37	3991	4656
38	DNR	DNR
39	3368	5132
40	1329	1946
Sum	14523	20815

VSNET Num.	Single Node Iterations	Mult. Node Iterations
1	1331	627
2	5094	2202
3	6891	4715
4	898	854
5	1058	1008
6	4827	3870
Sum	20099	13276

DNR: Did Not Run

TABLE B.4. DYNAMIC HEAP SORT MODIFICATION--RELAX-II(PQ)
The data in this table was generated by RELAX-II(PQ), a modification of RELAX-II that selects for each relaxation iteration a starting node with the maximum absolute deficit. Deficits for each node were maintained in a binary tree data structure (two-heap) which was maintained dynamically throughout the relaxation solution process.

NETGEN Num.	Sngle. Node Iterations	Mult. Node Iterations	Running Time
1	104	133	0.26
2	103	142	0.32
3	90	137	0.29
4	108	153	0.40
5	98	155	0.51
Sum	503	720	1.77
6	155	222	0.63
7	159	242	0.79
8	153	221	0.98
9	161	219	1.29
10	158	213	0.98
Sum	786	1117	4.66
11	904	66	1.09
12	363	82	2.30
13	423	94	5.18
14	229	92	5.29
15	181	92	5.29
Sum	2100	426	19.13
16	15	86	0.65
17	12	79	1.10
18	14	90	0.64
19	10	79	1.09
20	9	93	0.79
21	10	93	0.97
22	1	81	0.74
23	6	89	1.35
24	20	50	0.42
25	14	56	0.82
26	10	32	0.41
27	10	36	0.71
Sum	131	864	9.67

NETGEN Num.	Sngle. Node Iterations	Mult. Node Iterations	Running Time
28	99	125	1.26
29	90	118	1.76
30	90	121	2.32
31	76	127	1.71
32	184	195	2.76
33	134	191	2.44
34	121	198	2.59
35	156	193	3.51
Sum	950	950	18.35
36	374	1682	49.97
37	206	1290	28.86
38	DNR	DNR	DNR
39	22	1224	24.75
40	111	557	16.92
Sum	713	4753	120.50

VSNET Num.	Sngle. Node Iterations	Mult. Node Iterations	Running Time
1	531	479	3.23
2	2021	2799	24.91
3	1879	9119	79.32
4	81	192	5.26
5	163	226	19.92
6	384	1156	115.87
Sum	5059	13971	248.51

DNR: Did Not Run

TABLE B.5. SINGLE HEAP SORT MODIFICATION

This Table presents the runningt times obtained when RELAX-II was given only an initially sorted deficit list, in order of decreasing absolute node deficit, i.e., the sorted list of supply and demand nodes in decreasing absolute magnitude. Following the initial sort, no attempt was made to update the ordering of node deficits.

NETGEN Num.	Single Sort Time	Baseline RELAXII
1	0.17	0.18
2	0.25	0.26
3	0.28	0.26
4	0.41	0.48
5	0.28	0.33
Sum	1.40	1.51
6	0.50	0.48
7	0.54	0.61
8	0.60	0.71
9	0.84	0.63
10	0.62	0.71
Sum	3.10	3.14
11	0.15	0.12
12	0.17	0.14
13	0.27	0.24
14	0.27	0.23
15	0.46	0.45
Sum	1.33	1.19
16	0.30	0.26
17	0.41	0.37
18	0.33	0.31
19	0.58	0.34
20	0.39	0.34
21	0.57	0.43
22	0.37	0.32
23	0.54	0.60
24	0.23	0.18
25	0.50	0.46
26	0.21	0.13
27	0.40	0.26
Sum	4.82	4.01

NETGEN Num.	Single Sort Time	Baseline RELAXII
28	0.76	0.73
29	0.70	0.81
30	1.13	1.25
31	0.92	0.84
32	1.19	1.11
33	1.41	1.49
34	1.34	0.82
35	1.27	1.31
Sum	8.72	8.35
36	23.89	30.24
37	11.13	13.68
38	DNR	8.99
39	14.18	8.72
40	4.99	5.51
Sum	54.18	67.13

VSNET Num.	Single Node Iterations	Mult. Node Iterations
1	3.89	4.98
2	10.90	11.42
3	64.07	69.44
4	2.57	3.75
5	4.34	4.04
6	30.51	27.26
Sum	116.28	120.88

DNR: Did Not Run

TABLE B.6. PARTIAL SORT MODIFICATION--RELAX-II(PS)

This variation of RELAX-II compresses the DFCT array by removing all zeros. As node deficits become nonzero they are added to the end of DFCT, and as they become zero DFCT is compressed. A local search of breadth NTT is conducted within DFCT for each relaxation iteration to find the node with maximum absolute deficit. The optimal value of NTT changes with each problem solved, and there does not appear to be any way of pridicting its value ahead of time based on network characteristics. This Table contains the optimal NTT values for each NETGEN and VSNET problem, along with both RELAX-II(PS) and RELAX-II running times.

NETGEN Num.	NTT	Running Time	Baseline RELAXII
1	5	0.18	0.18
2	10	0.25	0.26
3	10	0.29	0.26
4	10	0.43	0.48
5	12	0.36	0.33
Sum		1.50	1.51
6	10	0.71	0.48
7	7	0.63	0.61
8	11	0.73	0.71
9	8	0.77	0.63
10	11	0.80	0.71
Sum		3.63	3.14
11	3	0.18	0.12
12	4	0.22	0.14
13	10	0.28	0.24
14	13	0.38	0.23
15	16	0.57	0.45
Sum		1.63	1.19
16	16	0.29	0.26
17	13	0.42	0.37
18	14	0.29	0.31
19	11	0.45	0.34
20	11	0.31	0.34
21	10	0.46	0.43
22	9	0.35	0.32
23	12	0.42	0.60
24	7	0.20	0.18
25	10	0.48	0.46
26	11	0.14	0.13
27	7	0.27	0.26
Sum		4.08	4.01

NETGEN Num.	NTT	Running Time	Baseline RELAXII
28	2	0.54	0.73
29	5	0.68	0.81
30	15	0.95	1.25
31	12	0.71	0.84
32	9	1.08	1.11
33	5	1.32	1.49
34	17	0.91	0.82
35	15	1.17	1.31
Sum		7.36	8.35
36	15	24.84	30.24
37	10	10.04	13.68
38	DNR	DNR	8.99
39	0	11.18	8.72
40	5	2.23	5.51
Sum		48.29	67.13

VSNET Num.	NTT	Running Time	Baseline VSNET
1	4	3.24	4.98
2	10	9.13	11.42
3	10	56.21	69.44
4	7	2.73	3.75
5	5	4.74	4.04
6	4	24.84	27.26
Sum		100.88	120.88

DNR: Did Not Run

LIST OF REFERENCES

Balas, E., and Hammer, P. L., "On the Transportation Problem--Part I," *Cahiers du Centre d'Etudes de Recherche Operationelle*, v. 4, p. 68, 1962.

Bertsekas, Dimitri P., "A Unified Framework for Primal-Dual Methods in Minimum Cost Network Flow Problems," *Mathematical Programming*, v. 32, pp. 125-145, 1985.

Bertsekas, Dimitri P., and Tseng, Paul, "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," *Operations Research*, v. 36, pp. 93-114, January 1988.

Bonwit, Willard R., *The Effect of Structure on the Solution Times of Minimum Cost Transportation and Multi-Echelon Network Flow Problems*, Master's Thesis, Navel Postgraduate School, Monterey, California, June 1984.

Bradley, Gordin H., Brown, Gerald G., and Graves, Glenn W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, v. 24, pp. 1-33, September 1977.

Dantzig, George B., "Application of the Simplex Method to a Transportation Problem," in *Activity Analysis of Production and Allocation*, T. C. Koopmans, ed., John Wiley & Sons, New York, 1951.

Fisher, Marshall L., "An Applications Oriented Guide to Lagrangian Relaxation," *Interfaces*, v. 15, pp. 10-21, March-April 1985.

Ford, L. R., and Fulkerson, D. R., "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," *Naval Research Logistics Quarterly*, v. 4, p. 47, March 1957.

Fulkerson, D. R., "An Out-of-Kilter Method for Minimal-Cost Flow Problems," *SIAM Journal of Applied Mathematics*, v. 9, p. 18, March 1961.

Grigoriadis, M. D., and Hsu, T., "The Rutgers Minimum Cost Network Flow Subroutines (RNET Documentation)," Department of Computer Sciences, Rutgers University, November 1980.

Harris, B., "A Code for the Transportation Problem of Linear Programming," *Journal of the Association for Computing Machinery*, v. 23, p. 155, January 1976.

Hitchcock, F. L., "The Distribution of a Product from Several Sources to Numerous Localities," *Journal of Mathematics and Physics*, v. 20, p. 224, 1941.

Klingman, D., Napier, A., and Stutz, J., "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation and Minimum Cost Flow Network Problems," *Management Science*, v. 20, pp. 814-821, January 1974.

Langley, R. W., Kennington, J., and Shetty, C. M., "Efficient Computational Devices for the Capacitated Transportation Problem," *Naval Research Logistics Quarterly*, v. 21, p. 637, December 1974.

Mulvey, J., "Column Weighing Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," Paper Presented at the Joint National Meeting of ORSA/TIMS, San Juan, Puerto Rico, Fall 1974.

Orden, A., "The Transshipment Problem," *Management Science*, v. 2, April 1956.

Rapp, Steven H., *Design and Implementation of a Network Optimizer for Officer Assignment During Mobilization*, Master's Thesis, Naval Postgraduate School, Monterey California, September 1987.

Rockfellar, R. T., *Network Flows and Monotropic Optimization*, John Wiley & Sons, New York, 1984.

Staniec, Cyrus James, *Design and Solution of an Ammunition Distribution Model by a Resource-Directive Multicommodity Network Flow Algorithm*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1984.

Tseng, Paul, *Relaxation Methods for Monotropic Programs*, Ph.D. Thesis, Massachusetts Institute of Technology, May 1986.

Yorio, Paul R., *A Spares Opgimization Model for Deployable U. S. Marine Corps Units*, Master's Thesis, Naval Postgraduate School, Monterey California, March 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Commandant of the Marine Corps Code TE 06 Headquarters, U. S. Marine Corps Washington D. C. 20380-0001	1
4.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
5.	Professor R. Kevin Wood, Code 55Wd Department of Operations Research Naval Postgraduate School Monterey, California 93943	10
6.	Professor Gerald G. Brown, Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93943	2
7.	Captain Michael B. Sagaser 4633 East Karen Drive Phoenix, Arizona 85032	2
8.	Chief of Naval Operations (OP-81) Department of the Navy Washington, D. C. 20350	1
9.	Dimitri P. Bertsekas and Paul Tseng Laboratory for Information and Decision Systems Massachusetts Institute of Technology Cambridge, Massachusetts 02139	1

Thesis
S152158 Sagaser
c.1 A computational comparison of the primal simplex and relaxation algorithms for solving minimum cost flow networks.